

C++ code

Name: Qasim amjad

Country: Pakistan

Date: 7/1/2021

Video library test

#include <gmock/gmock.h>

#include <gtest/gtest.h>

#include "../src/video.h"

using ::testing::ContainsRegex;

using ::testing::HasSubstr;

TEST(VideoLibrary, VideoLibraryHasAllVideos) {

VideoLibrary videoLibrary = VideoLibrary();

EXPECT\_EQ(videoLibrary.getVideos().size(), 5);

}

TEST(VideoLibrary, VideoLibraryParsesTagsCorrectly) {

std::vector<std::string> tags{"#cat", "#animal"};

VideoLibrary videoLibrary = VideoLibrary();

const Video \*video = videoLibrary.getVideo("amazing\_cats\_video\_id");

bool result = std::equal(tags.begin(), tags.end(), video->getTags().begin());

EXPECT\_EQ("Amazing Cats", video->getTitle());

EXPECT\_EQ("amazing\_cats\_video\_id", video->getVideoId());

EXPECT\_TRUE(result);

}

TEST(VideoLibrary, testLibraryParsesVideoCorrectlyWithoutTags) {

VideoLibrary videoLibrary = VideoLibrary();

const Video \*video = videoLibrary.getVideo("nothing\_video\_id");

EXPECT\_EQ("Video about nothing", video->getTitle());

EXPECT\_EQ("nothing\_video\_id", video->getVideoId());

EXPECT\_TRUE(video->getTags().empty());

}

**Part 1 test**

**#include <gmock/gmock.h>**

**#include <gtest/gtest.h>**

**#include "../src/videoplayer.h"**

**#include "../src/helper.h"**

**using ::testing::ContainsRegex;**

**using ::testing::HasSubstr;**

**TEST(Part1, numberOfVideos) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.numberOfVideos();**

**std::string output = testing::internal::GetCapturedStdout();**

**EXPECT\_THAT(output, HasSubstr("5 videos in the library"));**

**}**

**TEST(Part1, showAllVideos) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.showAllVideos();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 6);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here's a list of all available videos:"));**

**EXPECT\_THAT(commandOutput[1],**

**HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr("Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3],**

**HasSubstr("Funny Dogs (funny\_dogs\_video\_id) [#dog #animal]"));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr("Life at Google (life\_at\_google\_video\_id) [#google #career]"));**

**EXPECT\_THAT(commandOutput[5], HasSubstr("Video about nothing (nothing\_video\_id) []"));**

**}**

**TEST(Part1, playVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**}**

**TEST(Part1, playVideoNonExistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("some\_other\_video\_that\_doesnt\_exist");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Cannot play video: Video does not exist"));**

**}**

**TEST(Part1, playVideoStopPrevious) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.playVideo("funny\_dogs\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Playing video: Funny Dogs"));**

**}**

**TEST(Part1, playVideoDontStopPreviousIfNonExistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.playVideo("some\_other\_video");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0], Not(HasSubstr("Stopping video: Amazing Cats")));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Cannot play video: Video does not exist"));**

**}**

**TEST(Part1, stopVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.stopVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Stopping video: Amazing Cats"));**

**}**

**TEST(Part1, stopVideoTwice) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.stopVideo();**

**videoPlayer.stopVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2],**

**HasSubstr("Cannot stop video: No video is currently playing"));**

**}**

**TEST(Part1, stopVideoNothingPlaying) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.stopVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Cannot stop video: No video is currently playing"));**

**}**

**TEST(Part1, playRandomVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playRandomVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**ContainsRegex("Playing video: (Amazing Cats|Another Cat Video|Funny "**

**"Dogs|Life at Google|Video about nothing)"));**

**}**

**TEST(Part1, playRandomVideoStopsPreviousVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.playRandomVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**ContainsRegex("Playing video: (Amazing Cats|Another Cat Video|Funny "**

**"Dogs|Life at Google|Video about nothing)"));**

**}**

**TEST(Part1, showPlaying) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Currently playing: Amazing Cats "**

**"(amazing\_cats\_video\_id) [#cat #animal]"));**

**}**

**TEST(Part1, showNothingPlaying) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("No video is currently playing"));**

**}**

**TEST(Part1, pauseVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**}**

**TEST(Part1, pauseVideoShowVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2],**

**HasSubstr("Currently playing: Amazing Cats "**

**"(amazing\_cats\_video\_id) [#cat #animal] - PAUSED"));**

**}**

**TEST(Part1, pauseVideoPlayVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[4],**

**HasSubstr("Currently playing: Amazing Cats "**

**"(amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(output, Not(HasSubstr("PAUSED")));**

**}**

**TEST(Part1, pauseAlreadyPausedVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**videoPlayer.pauseVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Video already paused: Amazing Cats"));**

**}**

**TEST(Part1, pauseVideoNothingPlaying) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.pauseVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Cannot pause video: No video is currently playing"));**

**}**

**TEST(Part1, continueVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**videoPlayer.continueVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Continuing video: Amazing Cats"));**

**}**

**TEST(Part1, continueVideoNotPaused) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.continueVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Cannot continue video: Video is not paused"));**

**}**

**TEST(Part1, continueVideoNothingPlaying) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.continueVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr("Cannot continue video: No video is currently playing"));**

**}**

**Part 2 test**

**#include <gmock/gmock.h>**

**#include <gtest/gtest.h>**

**#include "../src/videoplayer.h"**

**#include "../src/helper.h"**

**using ::testing::HasSubstr;**

**using ::testing::MatchesRegex;**

**TEST(Part2, createPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**}**

**TEST(Part2, createExistingPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.createPlaylist("MY\_PLAYLIST");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Cannot create playlist: A playlist with the "**

**"same name already exists"));**

**}**

**TEST(Part2, addToPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_playLIST", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_playLIST: Amazing Cats"));**

**}**

**TEST(Part2, addToPlaylistAlreadyAdded) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr("Cannot add video to mY\_plaYList: Video already added"));**

**}**

**TEST(Part2, addVideoToPlaylistNonExistentVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "some\_other\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr("Cannot add video to mY\_plaYList: Video does not exist"));**

**}**

**TEST(Part2, addVideoToPlaylistNonExistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.addVideoToPlaylist("anotHER\_playlist", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Cannot add video to anotHER\_playlist: Playlist does not exist"));**

**}**

**TEST(Part2, addVideoToPlaylistNonExistentNoPlaylistNoVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.addVideoToPlaylist("anotHER\_playlist", "video\_does\_not\_exist");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Cannot add video to anotHER\_playlist: Playlist does not exist"));**

**}**

**TEST(Part2, showAllPlaylistsNoPlaylistsExist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.showAllPlaylists();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("No playlists exist yet"));**

**}**

**TEST(Part2, showAllPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.createPlaylist("anotHER\_playlist");**

**videoPlayer.showAllPlaylists();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Showing all playlists:"));**

**EXPECT\_THAT(commandOutput[4], HasSubstr("mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("anotHER\_playlist"));**

**}**

**TEST(Part2, showPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.showPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.showPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 6);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Showing playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("No videos here yet"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[4], HasSubstr("Showing playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[5],**

**HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**}**

**TEST(Part2, showPlaylistAfterRemoveAVideoFromPlaylistThenReAdd) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "life\_at\_google\_video\_id");**

**videoPlayer.removeFromPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.showPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 8);**

**EXPECT\_THAT(commandOutput[5], HasSubstr("Showing playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[6],**

**HasSubstr("Life at Google (life\_at\_google\_video\_id) [#google #career]"));**

**EXPECT\_THAT(commandOutput[7],**

**HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**}**

**TEST(Part2, showPlaylistNonExistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.showPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr("Cannot show playlist mY\_plaYList: Playlist does not exist"));**

**}**

**TEST(Part2, removeFromPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.removeFromPlaylist("MY\_playlist", "amazing\_cats\_video\_id");**

**videoPlayer.removeFromPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 4);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2],**

**HasSubstr("Removed video from MY\_playlist: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[3],**

**HasSubstr(**

**"Cannot remove video from mY\_plaYList: Video is not in playlist"));**

**}**

**TEST(Part2, removeFromPlaylistVideoNotInPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.removeFromPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(**

**commandOutput[1],**

**HasSubstr(**

**"Cannot remove video from mY\_plaYList: Video is not in playlist"));**

**}**

**TEST(Part2, removeFromPlaylistNonexistentVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.removeFromPlaylist("mY\_plaYList", "some\_other\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr("Cannot remove video from mY\_plaYList: Video does not exist"));**

**}**

**TEST(Part2, removeFromPlaylistNonexistentPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.removeFromPlaylist("my\_cool\_playlist", "some\_other\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Cannot remove video from my\_cool\_playlist: "**

**"Playlist does not exist"));**

**}**

**TEST(Part2, clearPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.addVideoToPlaylist("mY\_plaYList", "amazing\_cats\_video\_id");**

**videoPlayer.showPlaylist("mY\_plaYList");**

**videoPlayer.clearPlaylist("mY\_plaYList");**

**videoPlayer.showPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 7);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to mY\_plaYList: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Showing playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[3],**

**HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[4],**

**HasSubstr("Successfully removed all videos from mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[5], HasSubstr("Showing playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[6], HasSubstr("No videos here yet"));**

**}**

**TEST(Part2, clearPlaylistNonexistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.clearPlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr("Cannot clear playlist mY\_plaYList: Playlist does not exist"));**

**}**

**TEST(Part2, deletePlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("mY\_plaYList");**

**videoPlayer.deletePlaylist("MY\_PLAYLIST");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: mY\_plaYList"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Deleted playlist: MY\_PLAYLIST"));**

**}**

**TEST(Part2, deletePlaylistNonexistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.deletePlaylist("mY\_plaYList");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr("Cannot delete playlist mY\_plaYList: Playlist does not exist"));}**

**Part 3 test**

**#include <gmock/gmock.h>**

**#include <gtest/gtest.h>**

**#include "../src/videoplayer.h"**

**#include "../src/helper.h"**

**using ::testing::HasSubstr;**

**using ::testing::MatchesRegex;**

**TEST(Part3, searchVideosWithNoAnswer) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("No");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideos("cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(output, Not(HasSubstr("Playing video")));**

**}**

**TEST(Part3, searchVideosAndPlayAnswer) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("2");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideos("cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 6);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(commandOutput[5], HasSubstr("Playing video: Another Cat Video"));**

**}**

**TEST(Part3, searchVideosAnswerOutOfBounds) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("5");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideos("cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(output, Not(HasSubstr("Playing video")));**

**}**

**TEST(Part3, searchVideosInvalidNumber) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("ab3g");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideos("cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(output, Not(HasSubstr("Playing video")));**

**}**

**TEST(Part3, searchVideosNoResults) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.searchVideos("blah");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("No search results for blah"));**

**}**

**TEST(Part3, searchVideosWithTagNoAnswer) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("no");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideosWithTag("#cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for #cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(output, Not(HasSubstr("Playing video")));**

**}**

**TEST(Part3, searchVideosWithTagPlayAnswer) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("1");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideosWithTag("#cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 6);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for #cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(commandOutput[5], HasSubstr("Playing video: Amazing Cats"));**

**}**

**TEST(Part3, searchVideosWithTagAnswerOutOfBounds) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("5");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.searchVideosWithTag("#cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Here are the results for #cat:"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("1) Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("2) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**EXPECT\_THAT(output, Not(HasSubstr("Playing video")));**

**}**

**TEST(Part3, searchVideosWithTagNoResults) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.searchVideosWithTag("#blah");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("No search results for #blah"));**

**}**

**Part 4 test**

**#include <gmock/gmock.h>**

**#include <gtest/gtest.h>**

**#include "../src/videoplayer.h"**

**#include "../src/helper.h"**

**using ::testing::HasSubstr;**

**using ::testing::MatchesRegex;**

**TEST(Part4, flagVideoWithReason) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**}**

**TEST(Part4, flagVideoWithoutReason) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("another\_cat\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**EXPECT\_THAT(output, HasSubstr("Successfully flagged video: Another Cat Video "**

**"(reason: Not supplied)"));**

**}**

**TEST(Part4, flagVideoAlreadyFlagged) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(**

**commandOutput[1], HasSubstr("Cannot flag video: Video is already flagged"));**

**}**

**TEST(Part4, flagVideoNonexistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("video\_does\_not\_exist", "flagVideo\_reason");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(**

**commandOutput[0], HasSubstr("Cannot flag video: Video does not exist"));**

**}**

**TEST(Part4, flagVideoCanNoLongerPlay) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id");**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: Not supplied)"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Cannot play video: Video is currently flagged "**

**"(reason: Not supplied)"));**

**}**

**TEST(Part4, flagVideosPlayRandom) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("funny\_dogs\_video\_id");**

**videoPlayer.flagVideo("amazing\_cats\_video\_id");**

**videoPlayer.flagVideo("another\_cat\_video\_id");**

**videoPlayer.flagVideo("life\_at\_google\_video\_id");**

**videoPlayer.flagVideo("nothing\_video\_id");**

**videoPlayer.playRandomVideo();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 6);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Funny Dogs (reason: Not supplied)"));**

**EXPECT\_THAT(**

**commandOutput[1],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: Not supplied)"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr(**

**"Successfully flagged video: Another Cat Video (reason: Not supplied)"));**

**EXPECT\_THAT(**

**commandOutput[3],**

**HasSubstr(**

**"Successfully flagged video: Life at Google (reason: Not supplied)"));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"Successfully flagged video: Video about nothing (reason: Not supplied)"));**

**EXPECT\_THAT(commandOutput[5], HasSubstr("No videos available"));**

**}**

**TEST(Part4, flagVideoAddVideoToPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id");**

**videoPlayer.createPlaylist("my\_playlist");**

**videoPlayer.addVideoToPlaylist("my\_playlist", "amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: Not supplied)"));**

**EXPECT\_THAT(commandOutput[1],**

**HasSubstr("Successfully created new playlist: my\_playlist"));**

**EXPECT\_THAT(commandOutput[2],**

**HasSubstr("Cannot add video to my\_playlist: Video is "**

**"currently flagged (reason: Not supplied)"));**

**}**

**TEST(Part4, flagVideoShowPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("my\_playlist");**

**videoPlayer.addVideoToPlaylist("my\_playlist", "amazing\_cats\_video\_id");**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.showPlaylist("my\_playlist");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: my\_playlist"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to my\_playlist: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Showing playlist: my\_playlist"));**

**EXPECT\_THAT(commandOutput[4], HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat "**

**"#animal] - FLAGGED (reason: dont\_like\_cats)"));**

**}**

**TEST(Part4, flagVideoShowAllVideos) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.showAllVideos();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 7);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Here's a list of all available videos:"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat "**

**"#animal] - FLAGGED (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(**

**commandOutput[3],**

**HasSubstr("Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[4],**

**HasSubstr("Funny Dogs (funny\_dogs\_video\_id) [#dog #animal]"));**

**EXPECT\_THAT(**

**commandOutput[5],**

**HasSubstr("Life at Google (life\_at\_google\_video\_id) [#google #career]"));**

**EXPECT\_THAT(**

**commandOutput[6],**

**HasSubstr("Video about nothing (nothing\_video\_id) []"));**

**}**

**TEST(Part4, flagVideoSearchVideos) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("no");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.searchVideos("cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Here are the results for cat:"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("1) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**}**

**TEST(Part4, flagVideoSearchVideosWithTag) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**std::streambuf\* orig = std::cin.rdbuf();**

**std::istringstream input("no");**

**std::cin.rdbuf(input.rdbuf());**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.searchVideosWithTag("#cat");**

**std::cin.rdbuf(orig);**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Here are the results for #cat:"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("1) Another Cat Video (another\_cat\_video\_id) [#cat #animal]"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Would you like to play any of the above? If "**

**"yes, specify the number of the video."));**

**EXPECT\_THAT(**

**commandOutput[4],**

**HasSubstr(**

**"If your answer is not a valid number, we will assume it's a no."));**

**}**

**TEST(Part4, flagVideoStopPlayingVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 4);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("No video is currently playing"));**

**}**

**TEST(Part4, flagVideoStopPausedVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.pauseVideo();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 5);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Pausing video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[2], HasSubstr("Stopping video: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[3],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[4], HasSubstr("No video is currently playing"));**

**}**

**TEST(Part4, flagVideoKeepVideoPlayingIfDifferentFromFlaggedVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.playVideo("amazing\_cats\_video\_id");**

**videoPlayer.flagVideo("another\_cat\_video\_id", "dont\_like\_cats");**

**videoPlayer.showPlaying();**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 3);**

**EXPECT\_THAT(commandOutput[0], HasSubstr("Playing video: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[1],**

**HasSubstr(**

**"Successfully flagged video: Another Cat Video (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[2],**

**HasSubstr("Currently playing: Amazing Cats "**

**"(amazing\_cats\_video\_id) [#cat #animal]"));**

**}**

**TEST(Part4, allowVideo) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.allowVideo("amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 2);**

**EXPECT\_THAT(**

**commandOutput[0],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[1],**

**HasSubstr("Successfully removed flag from video: Amazing Cats"));**

**}**

**TEST(Part4, allowVideoNotFlagged) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.allowVideo("amazing\_cats\_video\_id");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Cannot remove flag from video: Video is not flagged"));**

**}**

**TEST(Part4, allowVideoNonexistent) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.allowVideo("video\_does\_not\_exist");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 1);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Cannot remove flag from video: Video does not exist"));**

**}**

**TEST(Part4, allowVideoShowPlaylist) {**

**VideoPlayer videoPlayer = VideoPlayer();**

**testing::internal::CaptureStdout();**

**videoPlayer.createPlaylist("my\_playlist");**

**videoPlayer.addVideoToPlaylist("my\_playlist", "amazing\_cats\_video\_id");**

**videoPlayer.flagVideo("amazing\_cats\_video\_id", "dont\_like\_cats");**

**videoPlayer.showPlaylist("my\_playlist");**

**videoPlayer.allowVideo("amazing\_cats\_video\_id");**

**videoPlayer.showPlaylist("my\_playlist");**

**std::string output = testing::internal::GetCapturedStdout();**

**std::vector<std::string> commandOutput = splitlines(output);**

**ASSERT\_EQ(commandOutput.size(), 8);**

**EXPECT\_THAT(commandOutput[0],**

**HasSubstr("Successfully created new playlist: my\_playlist"));**

**EXPECT\_THAT(commandOutput[1], HasSubstr("Added video to my\_playlist: Amazing Cats"));**

**EXPECT\_THAT(**

**commandOutput[2],**

**HasSubstr(**

**"Successfully flagged video: Amazing Cats (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[3], HasSubstr("Showing playlist: my\_playlist"));**

**EXPECT\_THAT(commandOutput[4], HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat "**

**"#animal] - FLAGGED (reason: dont\_like\_cats)"));**

**EXPECT\_THAT(commandOutput[5],**

**HasSubstr("Successfully removed flag from video: Amazing Cats"));**

**EXPECT\_THAT(commandOutput[7],**

**HasSubstr("Amazing Cats (amazing\_cats\_video\_id) [#cat #animal]"));**

**}**