**SOFTWARE DEVELOPMENT PRACTICES**

**STAFF GRADED ASSIGNMENT PART-2**

**LEKHSHREE PAUNIKAR 2022EBCS070**

Given below is a **Music Player** webpage that we will use as reference project to perform the two foundational DevOps practices: Version Control using Git and GitHub, and Containerization using Docker.



**Description of the Music Player Webpage (HTML, CSS, JavaScript):**

1. **Navigation Bar:** Contains the "Music Player" logo and title, placed horizontally at the top of page.
2. **Song List:** Displays a list of songs with cover images and song titles.
3. **Sticky Bottom Control Bar:** Contains control buttons (play/pause, next, previous) and a progress bar for the current song.

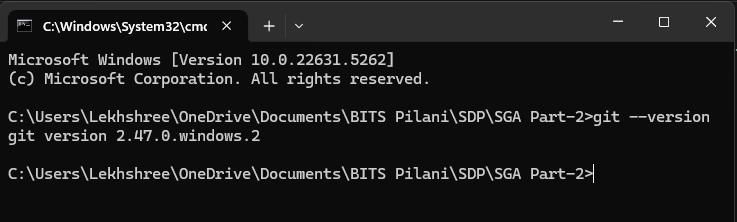
**Functionality:**

* Play/Pause: Toggles the audio playback state.
* Progress Bar: Shows the progress of the currently playing song and allows the user to seek through the track.
* Next/Previous Buttons: Navigate to the next or previous song in the playlist.

1. **GIF Animation**: A GIF is displayed when the song is playing, indicating the song's active state. Adds a visual cue that enhances the feedback of the user’s actions (play/pause).

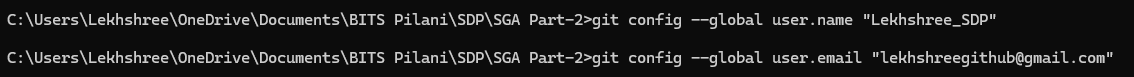
**QUESTION (1): Version Control using Git and GitHub.**

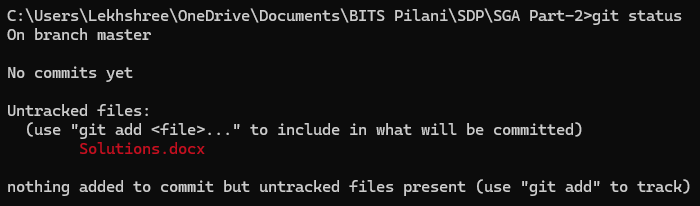
**GitHub link:** [**https://github.com/LekhshreePaunikar/SDP-SGA-2**](https://github.com/LekhshreePaunikar/SDP-SGA-2)

Checking the version of Git installed in the local system. git --version  
The output shows the currently installed version of Git in our local system.

Checking if the current folder is a Git Repository or not. git status  
  
The output suggest that the current folder is not a git repository. So before running any Git commands we first need to initialize it.

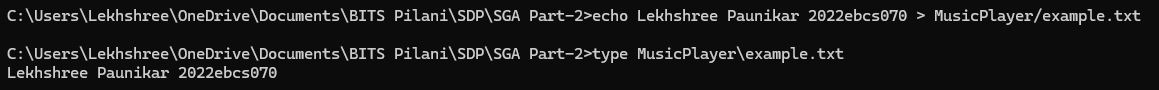
Initializing a new Git Repository. git init  
No errors suggest that folder is successfully initialized.

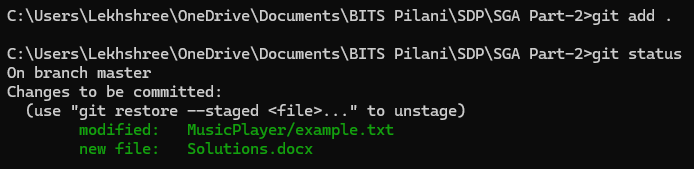
Setting up the Username and Email.  
git config --global user.name "Lekhshree\_SDP"  
git config --global user.email "lekhshreegithub@gmail.com"No errors suggest that username and email are successfully set up.

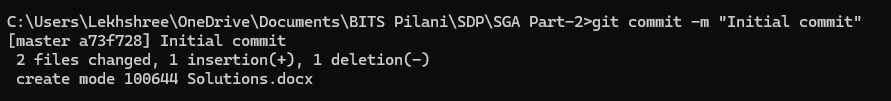
Checking the Git status. git status  
  
It shows that a file “Solutions.docx” is untracked file. This is an empty Word Document file meant to store the solutions of the **SDP SGA-2 Assignment**. Though it exists in the directory, but it has not been added to Git for version control yet.

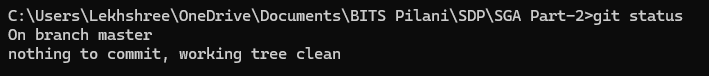
Creating a new project folder named “MusicPlayer” inside which we will write all of our code (HTML, CSS and Javascript). mkdir MusicPlayer  
No errors suggest successful creation of folder.

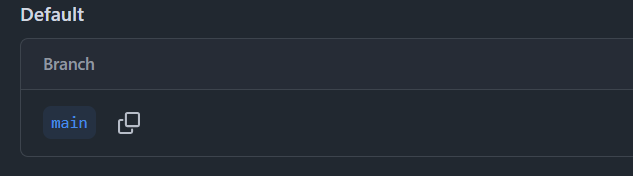
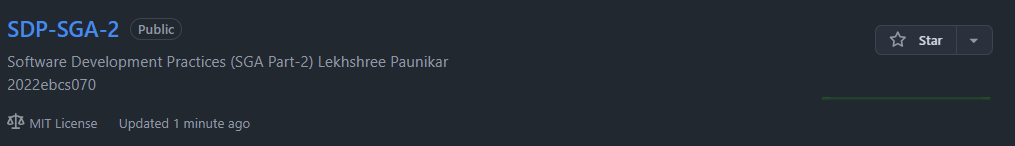
Adding the newly created folder to Git. git add MusicPlayer  
Since Git does not track empty directories, so we must make sure that there is at least one file inside the folder. So, we will make a text file named “example.txt” containing my student ID.  
echo Lekhshree Paunikar 2022ebcs070 > MusicPlayer/example.txt

Checking the content of example.txt file. type MusicPlayer\example.txt  
No errors suggest successful creation of example.txt file.

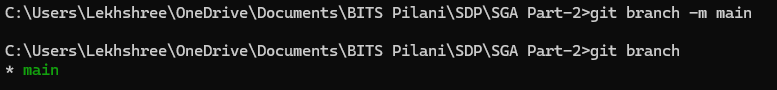
Adding all files in the current directory and subdirectories to the Git staging area.   
git add .  
git status  
  
The green colored files are the ones that are Staged. After staging the files, we need to commit them. We will save the changes with a message "Initial commit."

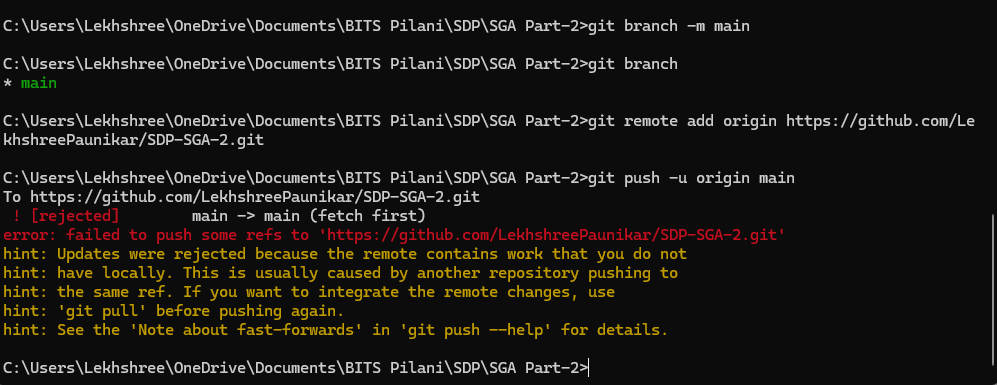
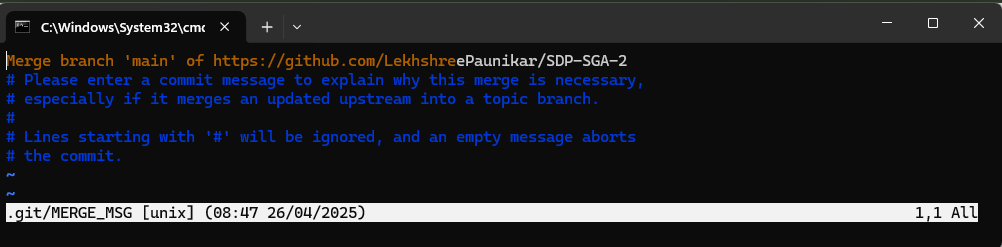
Making the first commit using Git. git commit -m "Initial commit"  
No errors suggest successful commit.

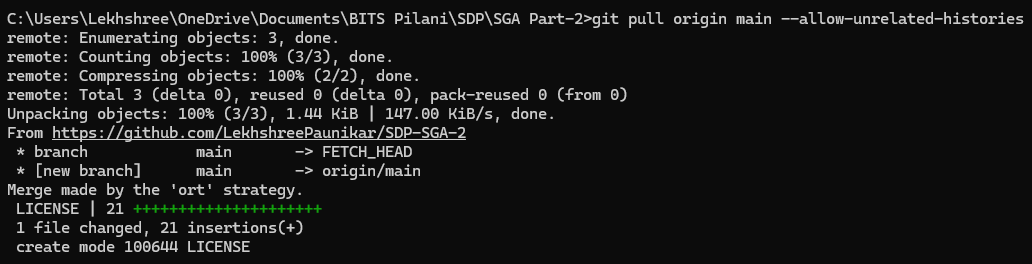
Checking the committed changes again using git status. git status  
The above screen shot verifies that master branch is clean.

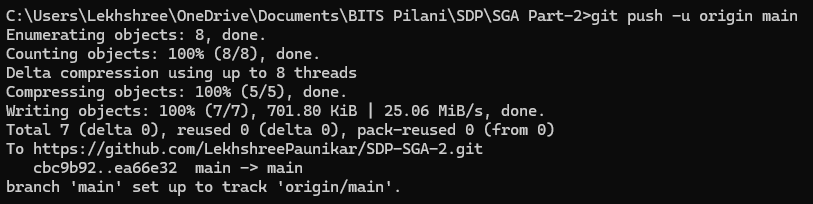
Now we have to connect this local repository to a remote **GitHub repository**, for which we will create a new repository in our GitHub Account. I prefer not to initialize a README.md file and use the MIT license for the new Repository.   
The above screenshots suggest successful creation of repository on GitHub. We can see that it’s default branch “main” has been automatically created.

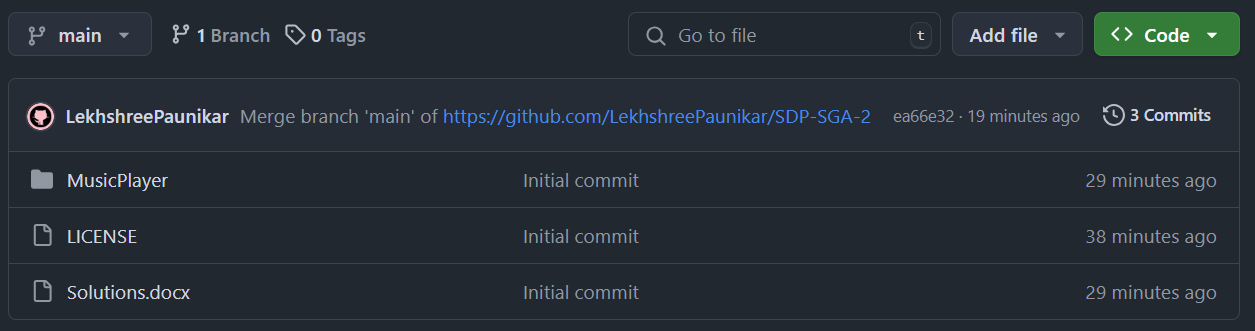
***Resolving Conflicts:*** *Since we are on master branch locally because Git initializes a new repository with the default branch name as master. But on GitHub, our default branch is main, as seen in the screenshot. If we want our local branch name (master) to match GitHub’s (main) we have to first rename it and then connect and push it to GitHub.*

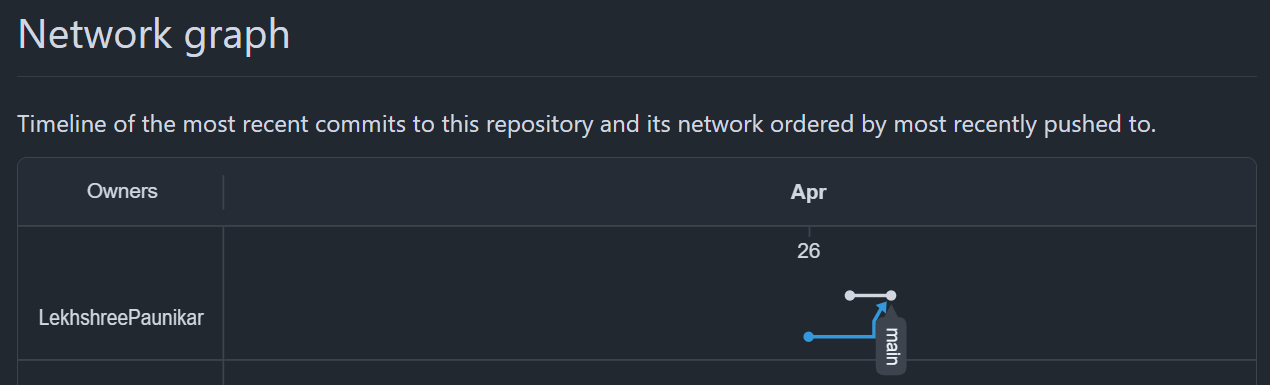
Renaming the branch. git branch -m main  
No errors suggest successful renaming of branch.

Adding the Remote Repository URL.  
git remote add origin <https://github.com/LekhshreePaunikar/SDP-SGA-2.git>  
git push -u origin main  
***Resolving Errors****: This error is shown in terminal because our GitHub repo (main) already has some commits, while local repo's main has no knowledge of that, so Git is preventing a direct push to avoid overwriting changes. So, we first have to pull from GitHub so our local branch syncs with the remote. We will use a special tag “--allow-unrelated-histories” so that Git sees our local and remote histories as unrelated.*   
git pull origin main --allow-unrelated-histories  
Running the above command will open the Vim editor where we have to type the commit message. Since current message is fine, we will not change this default message and save-exit Vim editor.  
Successfully exited the Vim editor.

  
Successfully pulled origin from GitHub

  
Successfully pushed changes to GitHub.

  
Screen shot of GitHub validating the successful push from Terminal.

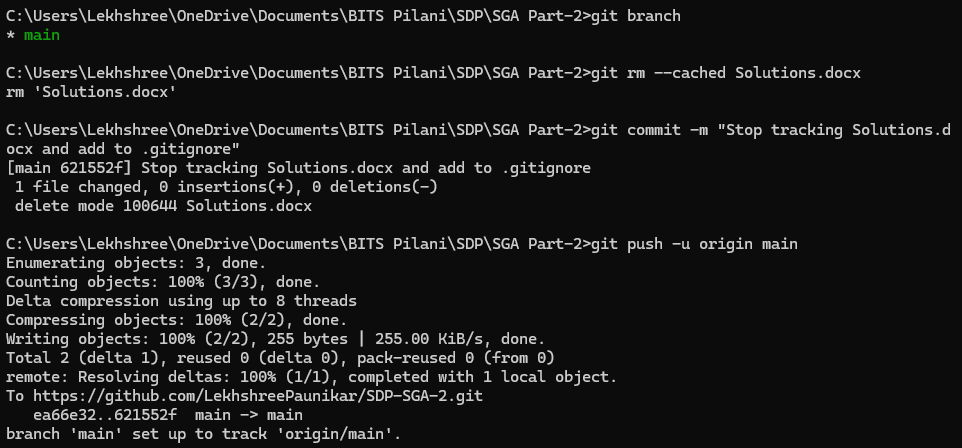
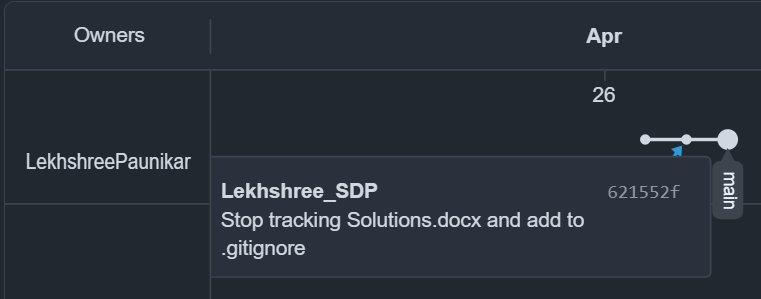
  
This screen shot shows the current state of the Network graph. We can see that vertical blue line on April 26 means our local and remote main branches have merged successfully. The tag main on the latest commit confirms that we are currently on the main branch, and it is active. The grey node pointing to the same spot as blue means GitHub has the same latest commit.

***Making the workflow easier:*** *Since I am continuously making changes in Solutions.docx file. I want Git to ignore it and instead track other important files. Since Solutions.docx is already tracked and committed, simply adding it to* ***.gitignore*** *won’t untrack it. We need to untrack it manually.*

At the root of our project folder, we will create a .gitignore file and add Solutions.docx to it which will Git to ignore future changes to this file.

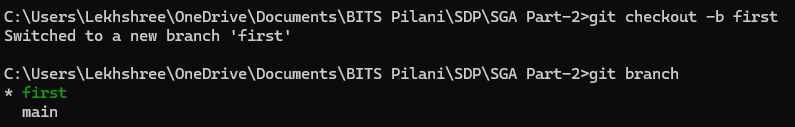
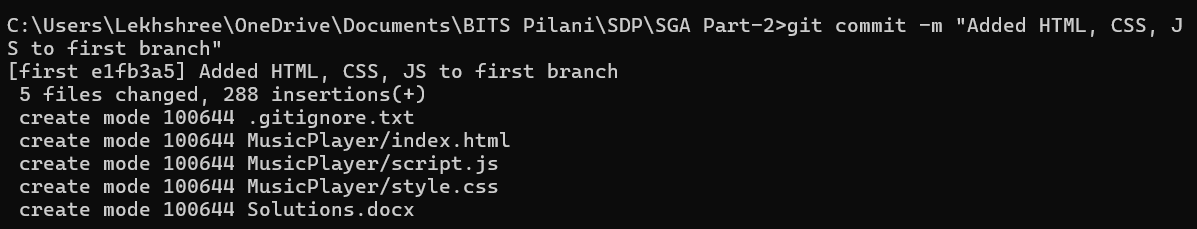
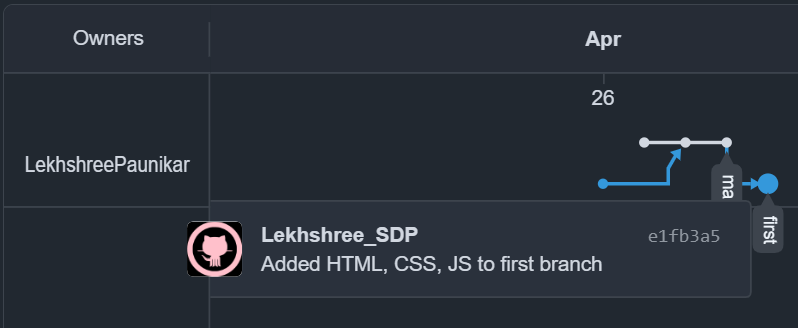
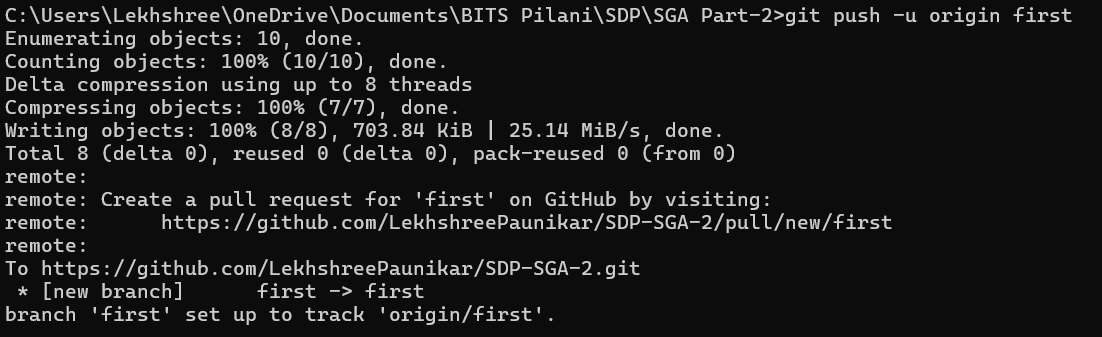
Untrack the file using Command: git rm --cached Solutions.docx ,where “—cached” tag removes it only from Git, not from our local disk.

Then commit and push the changes to the main branch.  
git commit -m "Stop tracking Solutions.docx and add to .gitignore"  
git push -u origin main

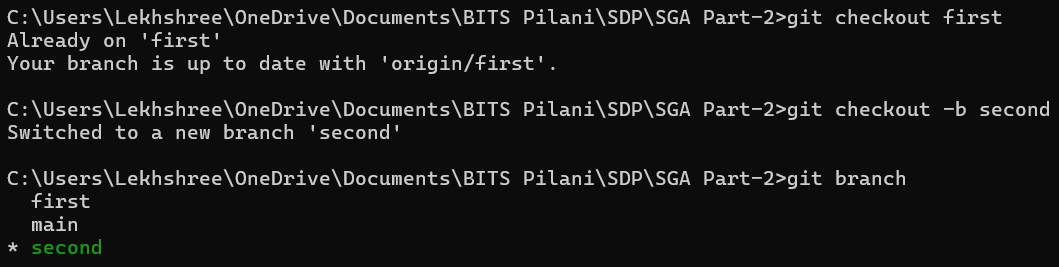
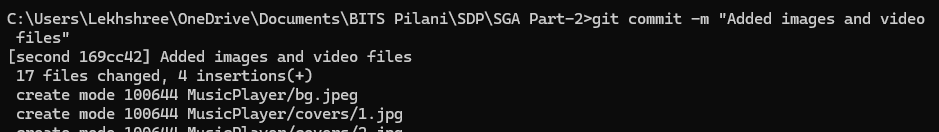
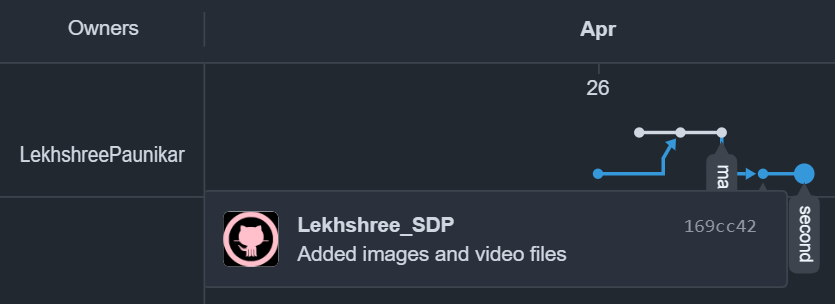
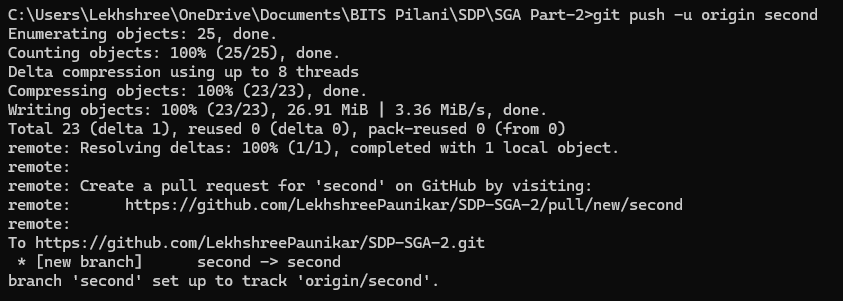
***Personal Learning:*** *We only needed “-u” tag used for “r --set-upstream” only once to link our local main to GitHub’s origin/main. But since I fumbled multiple times here and did not want to risk it again. So, I checked the current branch and again used complete push-upstream command instead of simple “git push”.*  
  
Successfully stopped Git from tracking Solutions.docx file.  
  
  
The above screenshot of Network Graph confirms successful untracking.

**Branching using Git and GitHub:**

Now we will create a branch named “**first**” originating from “**main**” and make some changes in it.  
git checkout -b first  
git add .  
git commit -m "Added HTML, CSS, JS to first branch"  
git push -u origin first

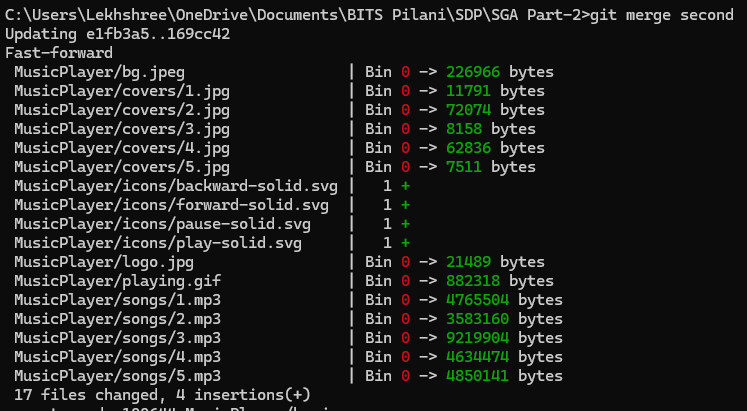
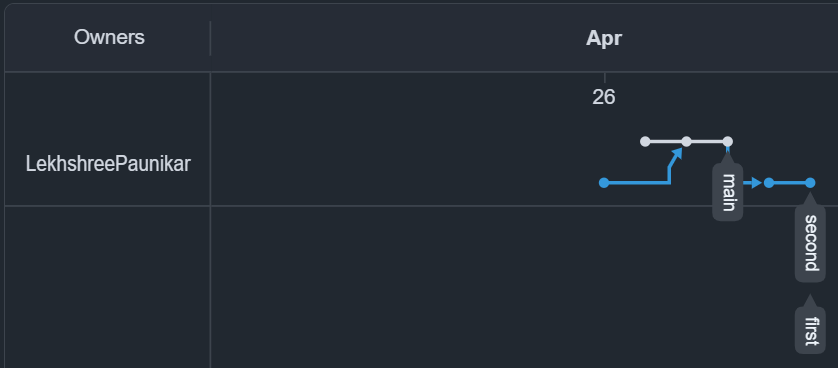
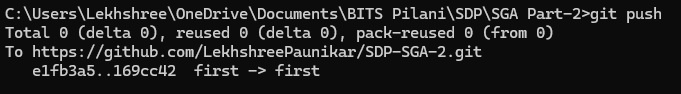
  
Changed and verified the current branch name (first).  
Added all changes.  
Committed all changes.  
The above screen shots from terminal and of Network Graph Verifies Successful creation of “first” branch and commits made to it.

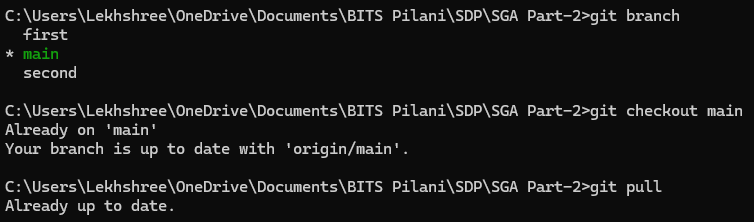
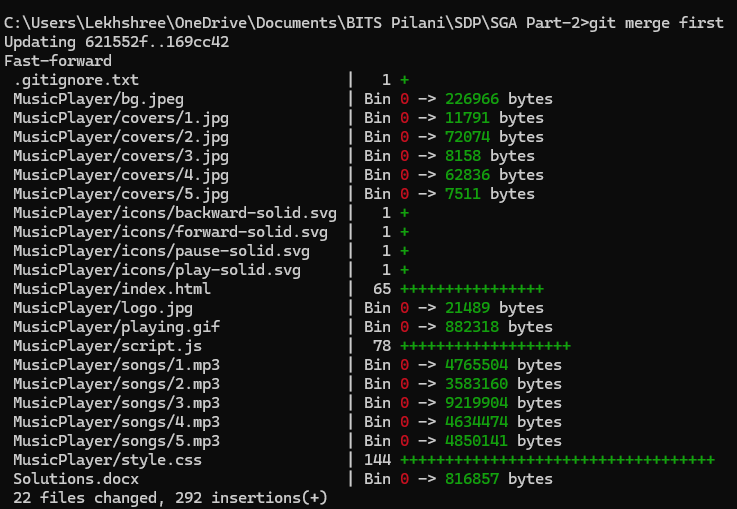
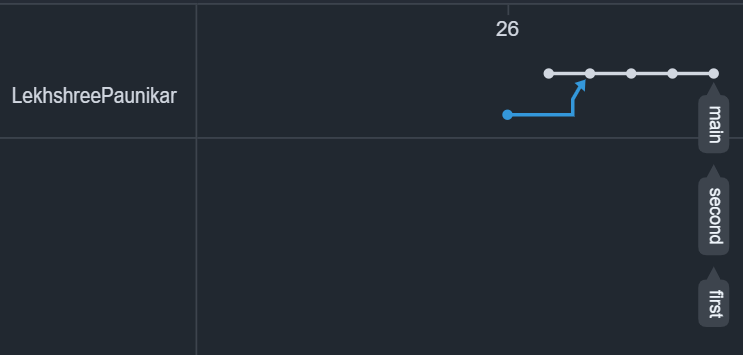
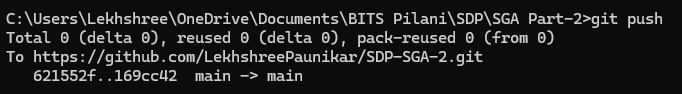
Now we will create a new branch named “**second**” originating from “**first**” and make changes in it.  
git checkout first  
git checkout -b second  
git add .  
git commit -m "Added images and video files"  
git push -u origin second

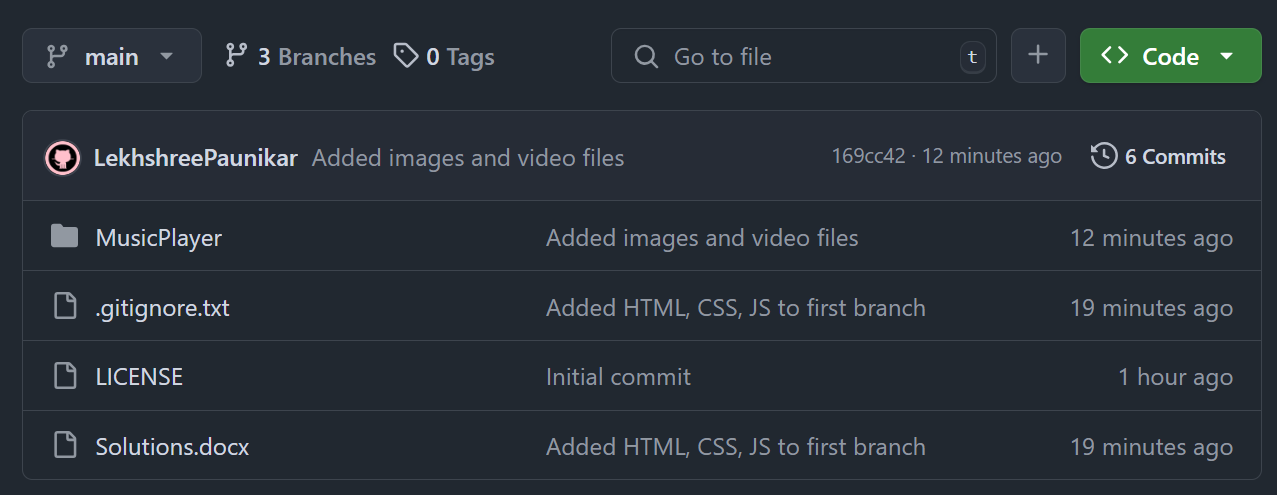
  
Successfully created and changed the branch to “second”.  
Added changes to it.  
Committed the changes.  
The above screen shots from terminal and of Network Graph verifies Successful creation of “second” branch and commits made to it.

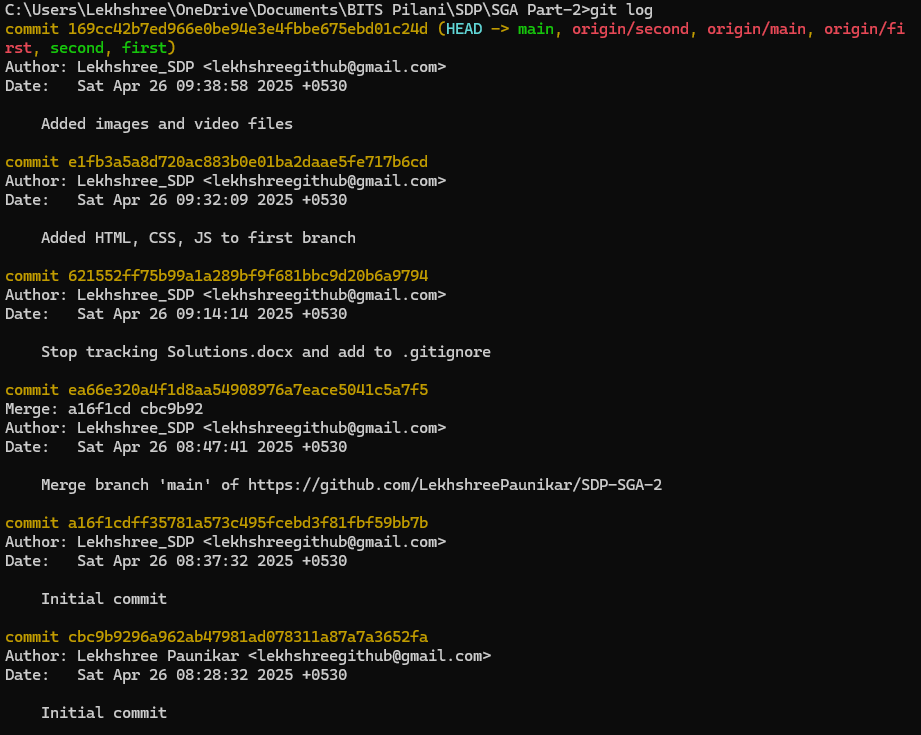
**Merging of Branches:**

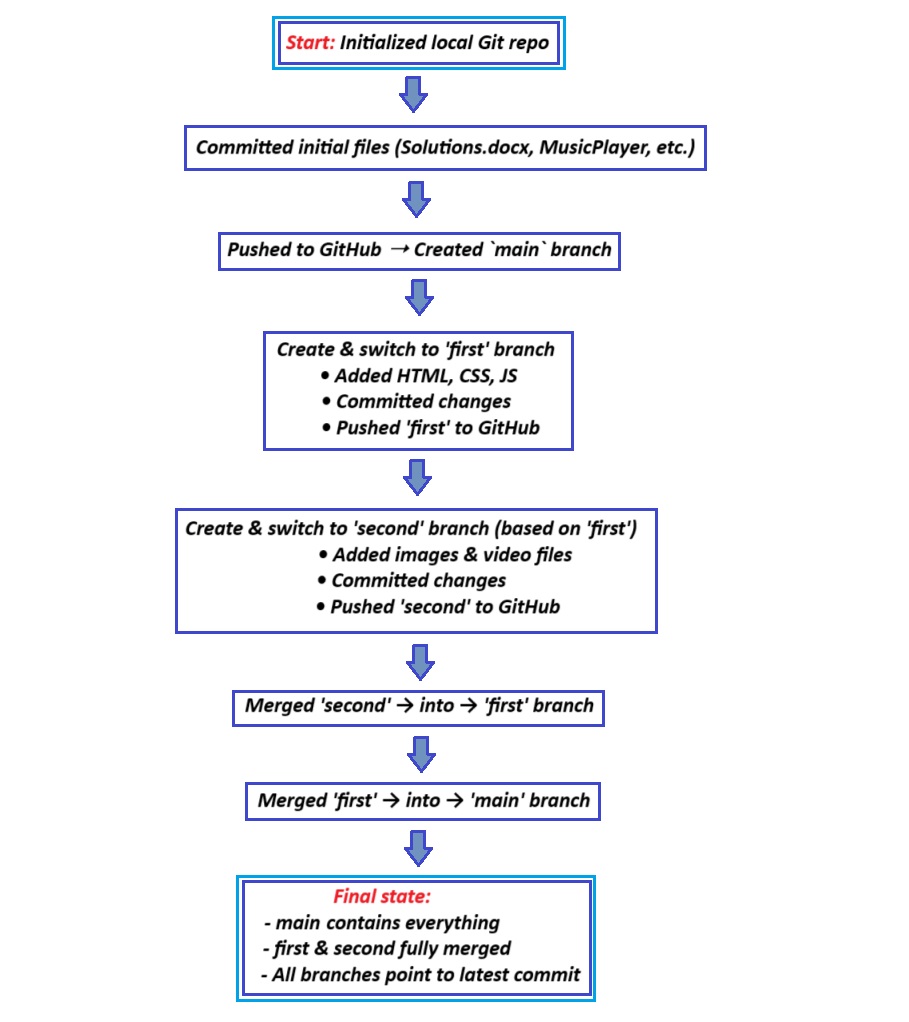
Now we will merge all the branches in clean and logical branching sequence. First, we will merge "**second**" branch into "**first**" branch, and then we will merge "first" branch back into "**main**" branch this way, main will contain files from both the branches without any conflicts.   
git checkout first  
git pull  
git merge second  
git push

  
Checking the current state of “first” branch.  
Successfully merged "second" branch into "first" branch.  
The above screenshot of terminal and Network Graph verifies successful merger and push.

Merging "first" branch back into "main" branch.  
git checkout main  
git pull  
git merge first  
git push  
  
Checking the current branch name and its current state.  
Successful merger of "first" branch into "main" branch.  
The above screenshot of terminal and Network Graph verifies successful merger and push.

  
This is the latest state of GitHub Repository.   
***Note:*** ***Since file “Solutions.docx”(untracked) was included in .gitignore file. Its content is still raw and unrefined, so I will remove it from my GitHub repository and present the final Solutions.pdf in the ZIP file along with other code files.***

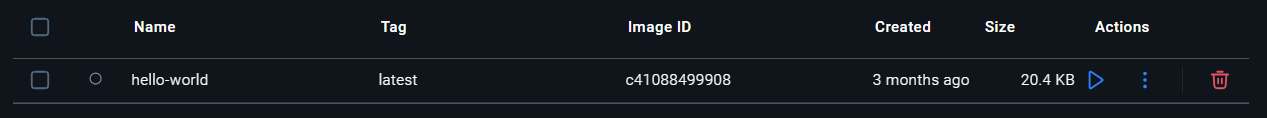
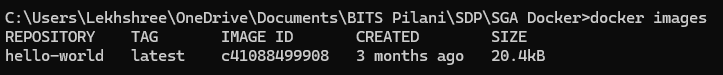
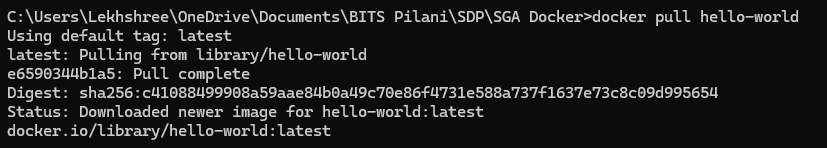
We can verify all the changes made up till now by reading the Logs.   
**Command: git log**   
The screen shot of the Terminal given below verifies all the Commits we have done up till now.

**The flowchart given below diagrammatically explains and summarizes all the git logs:**  
  


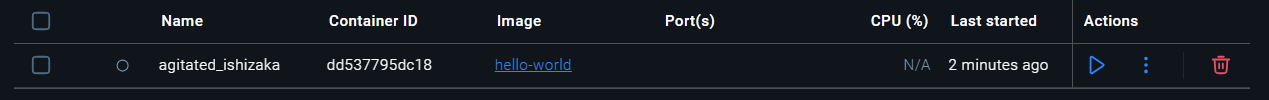
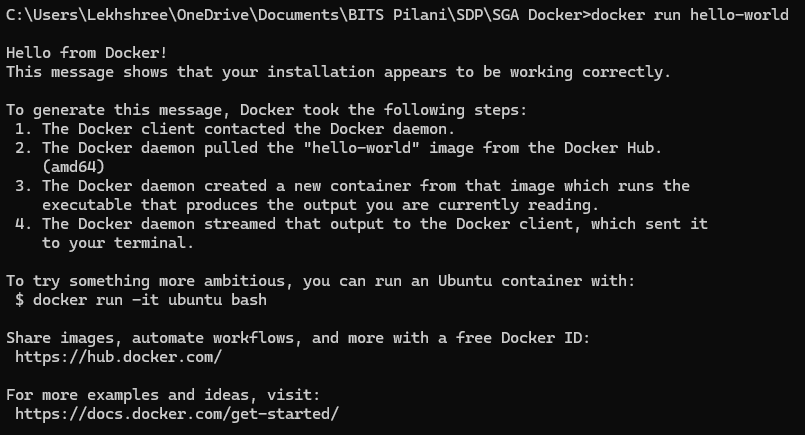
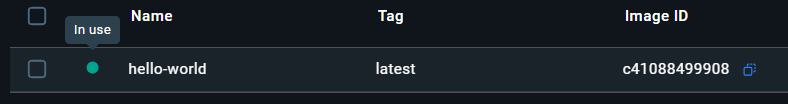
**QUESTION (2): Containerization using Docker**

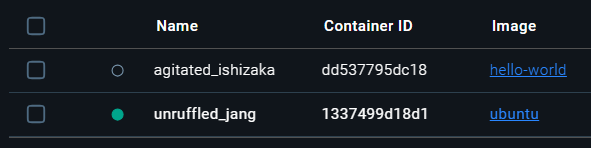
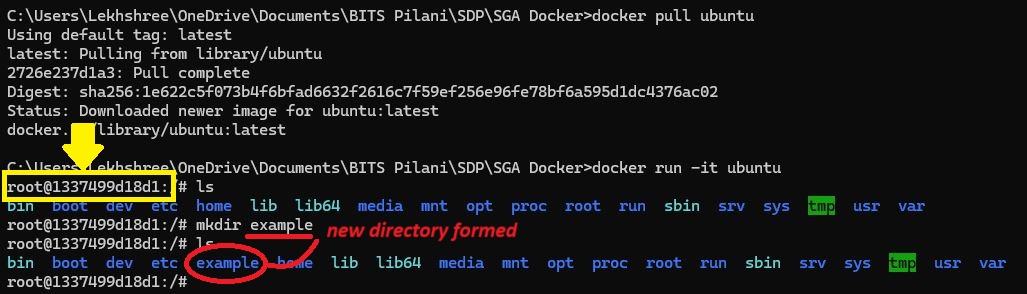
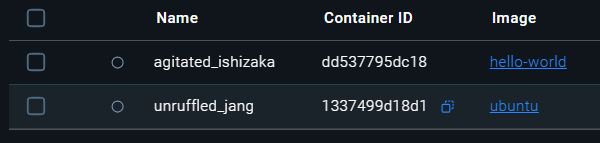
**Link:** [**https://hub.docker.com/r/lekhshreedocker/musicplayer-app**](https://hub.docker.com/r/lekhshreedocker/musicplayer-app)

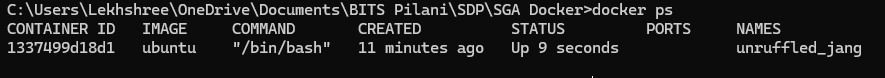
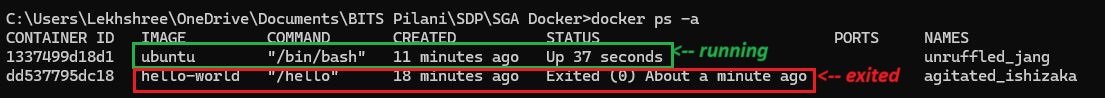
To use “docker” in the terminal we first have to install “Docker.Desktop” and log in using valid credentials. We can check the version of docker installed in local machine using command: docker -v   
The above screen shot shows successful installation of docker desktop on windows.

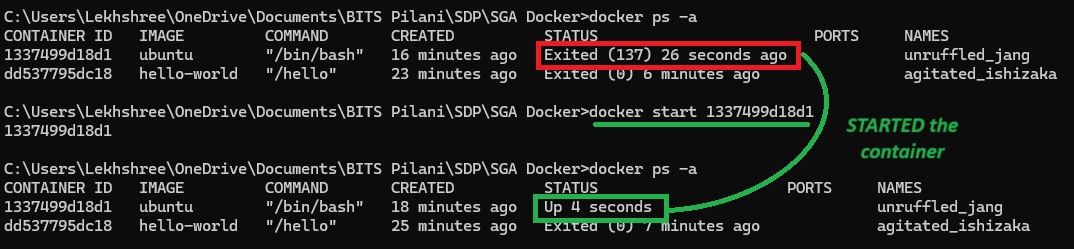
We can pull a Docker Official Image “hello-world” with 1B+ downloads using command “docker pull hello-world” in the command prompt of our project folder. And check our recently pulled docker image using “docker images”. This image can also be seen on our docker desktop.    
The above screen shots confirm successful pulling of docker image in the local machine.

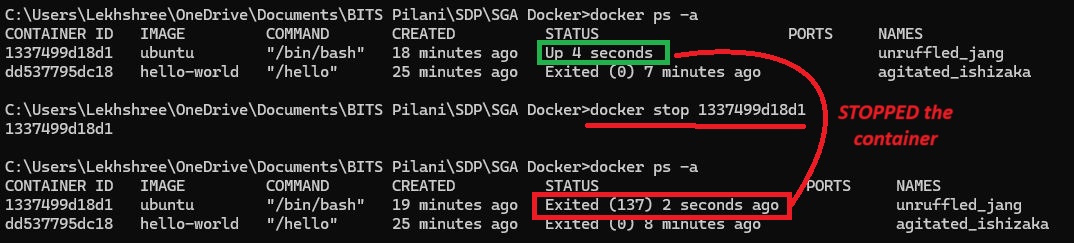
***Learning from observations:*** *As we can see, the size of this image is in kilobytes (very small). This confirms that* ***Docker is a lightweight technology compared to using traditional virtual machines (VMs).*** *Unlike VMs, which virtualize entire hardware and require a full guest OS, Docker containers share the host system's kernel and isolate only the application layer. This efficient model justifies the small size of Docker images.*

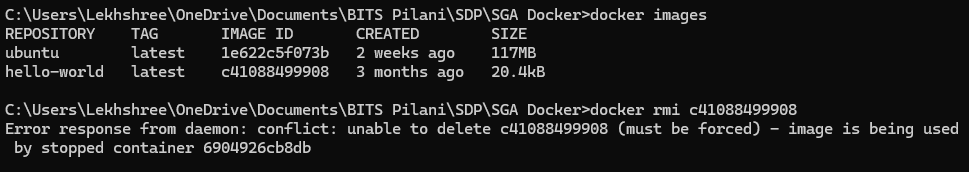
Now we can create a container using this image. Command: docker run hello-world  
The above screen shots confirm successful creation of container using image.  
In the above screen shot we can see a “green dot” against docker image, it signifies that we have just used this image to create the container.

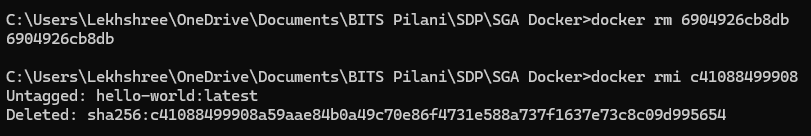
We can also create a new container and run it in interactive-terminal mode using “-it” tag.  
docker pull ubuntu  
docker run -t ubuntu  
From the above images we can see that the root@1337499d18d1 matches the ContinerID which we created using interactive made. Geen dot confirms that this container is currently in running state. If we “exit”, the green dot will fade away, indicating its in exited state.  
   
The above screen shots show container in “exit” state.

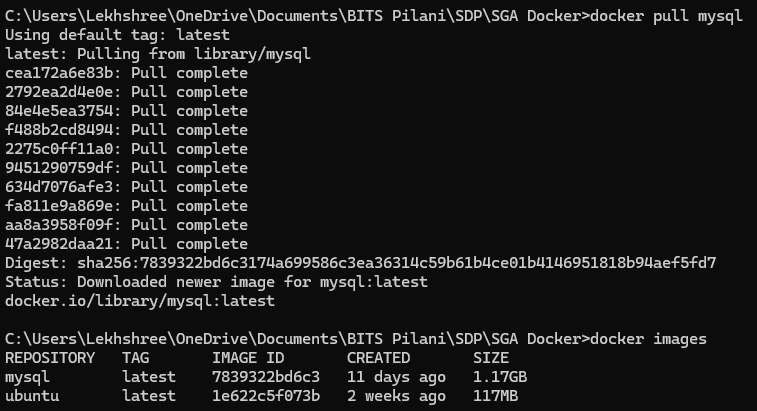
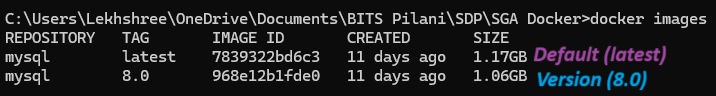
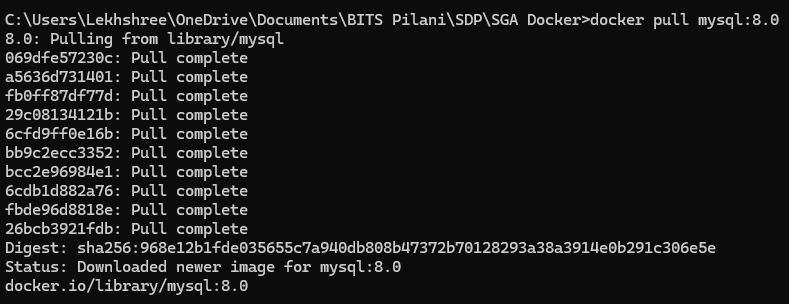
We can see the curently running container using “docker ps”, by adding tag “-a” we can see both running and exited containers.  
Command: docker ps  
Command: docker ps -a

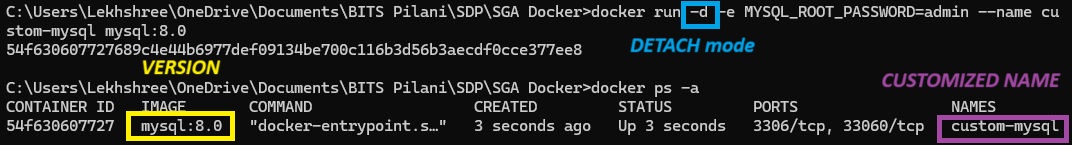
Command “docker run <image\_name>” creates a new container from the same image everytime it is run on terminal. In order to start previously created container we can use command “docker start” command followed by either <container\_name> or <container\_ID>.   
docker start 1337499d18d1 or docker start unruffled\_jang

Similarly we can two versions of same commad to stop the running container “docker stop” followed by either <container\_name> or <container\_ID>.  
docker stop 1337499d18d1 or docker stop unruffled\_jang

To remove “hello-world” image from the loacl machine we use command “docker rmi” followed by either <image\_name> or <image\_ID>.  
docker rmi c41088499908 or docker rmi hello-world  
The above screen shots shows an error because we already have a continer made from this image.   
***Resolving Error:*** *In order to remove this image we first have to remove the container.*

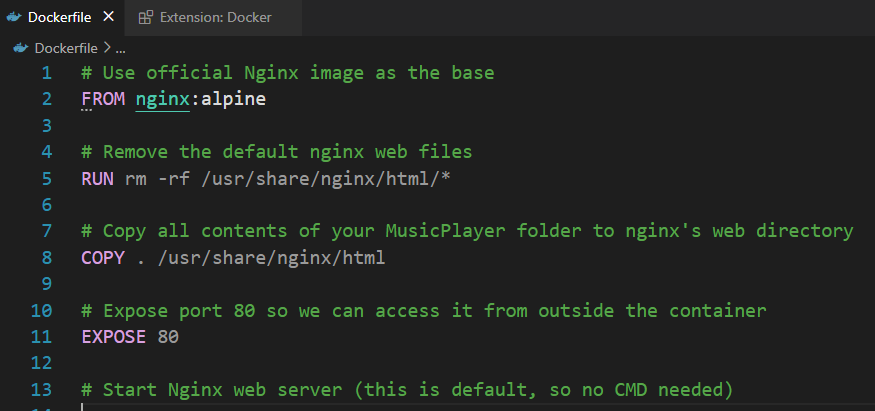
For this we will use command use command “docker rm” followed by either <container\_name> or <container\_ID>.  
docker rm 6904926cb8db or docker rm nifty\_brahmaguptaThe above screen shot shows successful removal of docker container and its image.   
***Note:*** *The name and id of container have changed because I had made an error and had to start again from pulling the image step.*

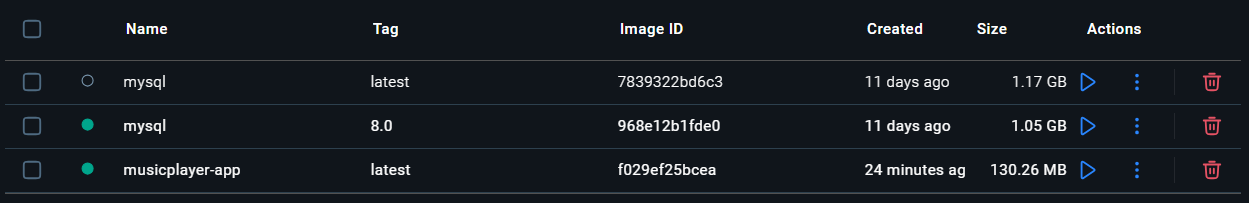
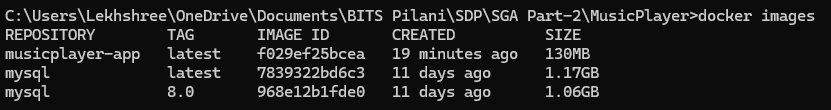
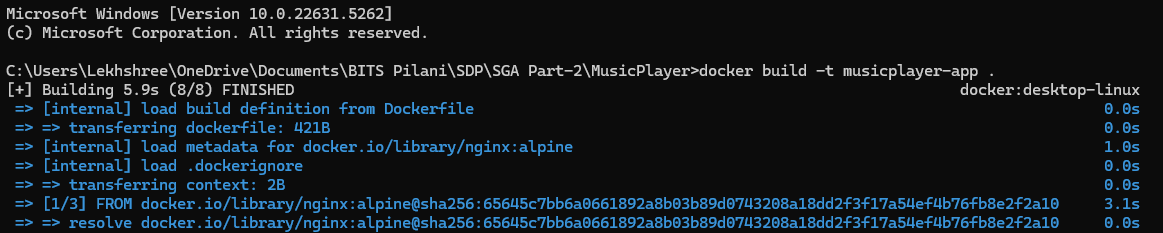
We can pull a customized mysql image (with different version) from dockerhub. For which we will first create latest(default) image and then the customized image.   
Command for defualt version: docker pull mysql  
   
Command for specified version: docker pull mysql:8.0  
The above screenshot shows the presence of both versions of mysql image: default(latest) and 8.0.

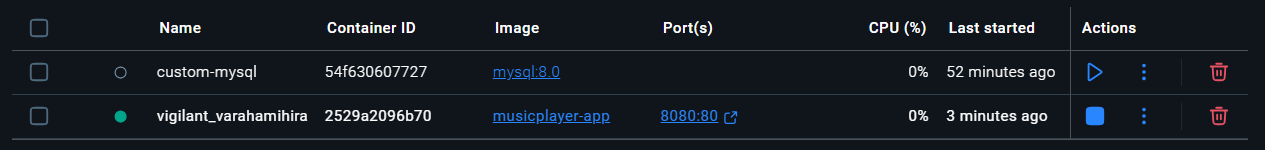
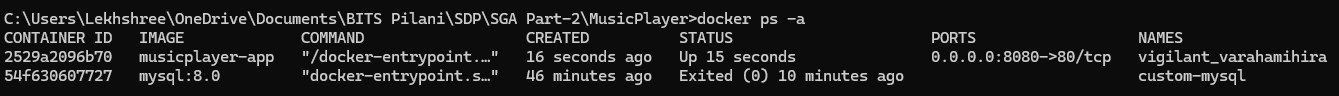
Similary, we can also customized container using tag “-d” for detach mode, tag “--name” for naming and “mysql:8.0” to specifically use the version 8.0 of mysql image.   
Command: docker run -d -e MYSQL\_ROOT\_PASSWORD=admin --name custom-mysql mysql  
The above screen shot shows successful creation of customized docker container.  
***Note:*** *we have used additional tag “-e” to set the environment variable for “MYSQL\_ROOT\_PASSWORD= admin” as it required during the creation of mysql container.*

**Containerizing MusicPlayer Website project:**

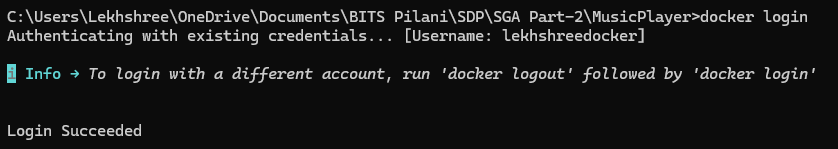
Now we will containerize the “**MusicPlayer**” website. For this, we first have to write **Dockerfile** script in side the same folder as “**index.html**” because we will be using **Nginx** to host our static website. Nginx automatically looks for an index.html file inside **/usr/share/nginx/html** and serves it. Also we need to add **Docker extension** in the VSCode.

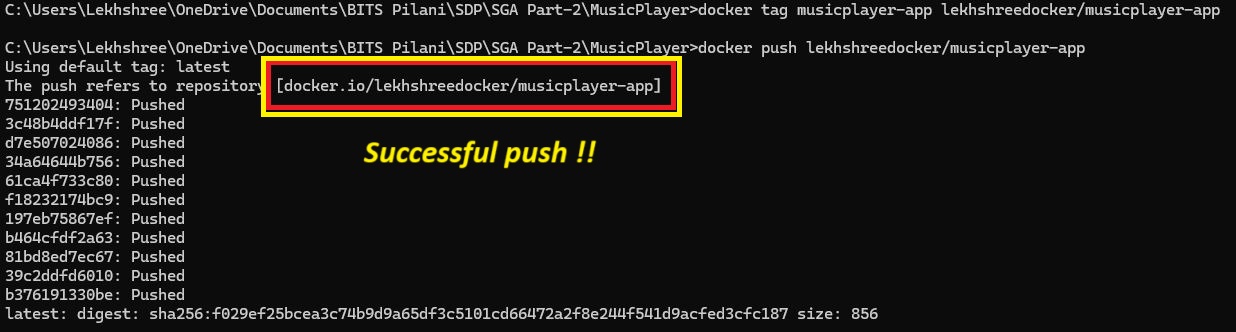
**The script of Dockerfile is as follows:**  
***Explanation:*** This Dockerfile uses a lightweight **Nginx server** (nginx:alpine) to host MusicPlayer web project. It removes Nginx's default files, copies custom static files (HTML, CSS, JS, images, mp3s) into the server, and exposes **port 80** for web access. No additional configuration is needed as the site will run immediately inside the Docker container.

We can use this Dockerfile to build the docker image named “musicplayer-app”.  
Command: docker build -t musicplayer-app .  
The above screen shots from terminal and DockerDesktop shows successful creation of docker image.

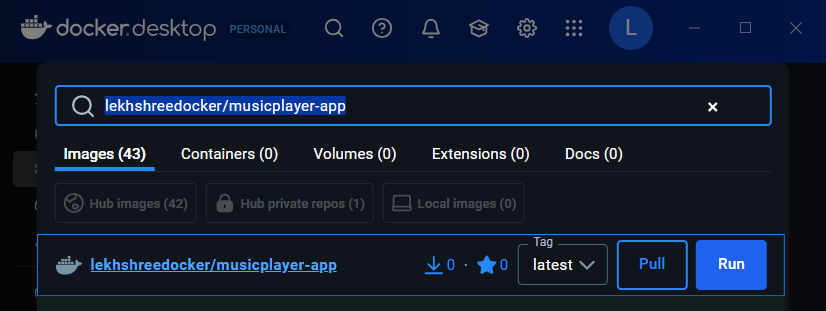
We can run a docker container in detached mode and map port 80 to localhost:8080 (<http://localhost:8080>).  
Command: docker run -d -p 8080:80 musicplayer-app  
The above screen shots from terminal and DockerDesktop shows successful creation of docker container.

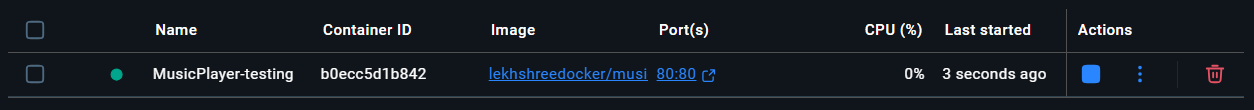
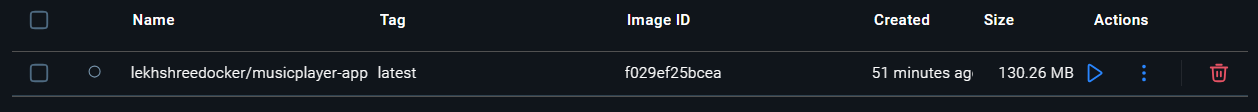
  
The above screen shot is from “<http://localhost:8080>”, verifying successful containerization of MusicPlayer project.

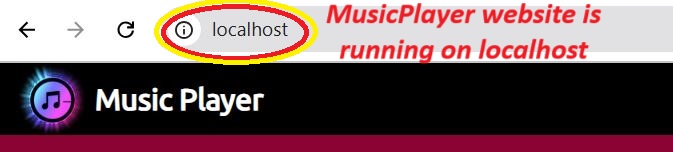
We can push Docker image “musicplayer-app” to DockerHub, for which we first need to login to Docker in the terminal. Command: docker login  
The screen shot shows successful login.

Before pushing the Docker Image we first need to tag it with DockerHub username “lekhshreedocker”.  
Command for tagging: docker tag musicplayer-app lekhshreedocker/musicplayer-app  
Command for pushing: docker push lekhshreedocker/musicplayer-app  
The above screen shot shows successful push of docker image to DockerHub.

After successful push, it will appear in the Repositories section of DockerHub profile.  
**Link:** [**https://hub.docker.com/r/lekhshreedocker/musicplayer-app**](https://hub.docker.com/r/lekhshreedocker/musicplayer-app)

We can double verify this image by pulling it back in DockerDesktop and running it on a different port.  
In the above screen shot we are trying to pull the image using DockerDesktop’s search bar.

 In the above screen shots we can see that we have successfully pulled the docker image and made a running container using it.

  
The above screen shot is from “<http://localhost:80>”, verifying successful containerization and running of MusicPlayer project using Docker.