

**Table of Contents**

[**Git** 1](#_Toc108765934)

[**Testing** 4](#_Toc108765935)

[**TDD** 4](#_Toc108765936)

[**BDD** 10](#_Toc108765937)

[**Conclusion** 16](#_Toc108765938)

[**Links** 17](#_Toc108765939)

[**GitHub:** 17](#_Toc108765940)

[**YouTube:** 17](#_Toc108765941)

[**References** 18](#_Toc108765942)

**Table of Figures**

[Figure 1: Working on a branch with GitHub Desktop 2](#_Toc108599903)

[Figure 2: Album model insert and delete test 5](#_Toc108599904)

[Figure 3: Featured playlist model insert and update test 6](#_Toc108599905)

[Figure 4: Featured Playlist Song insert and delete test 7](#_Toc108599906)

[Figure 5: Like model insert and delete test 8](#_Toc108599907)

[Figure 6: Report insert and delete test 9](#_Toc108599908)

[Figure 7: All TDD tests completed successfully 10](#_Toc108599909)

[Figure 8: Edit Song class test 11](#_Toc108599910)

[Figure 9: Password Setting class test 12](#_Toc108599911)

[Figure 10: Sign Up class test 13](#_Toc108599912)

[Figure 11: Upload Song class test 14](#_Toc108599913)

[Figure 12: Upload Album class test 15](#_Toc108599914)

[Figure 13: All BDD tests completed successfully 15](#_Toc108599915)

# **Git**

There are a total of five distinct working methods on GitHub, all of which are categorized as basic git workflows, where there is only one master branch and all other developers work on the same branch. This method is uncomfortable since it seems like there are many potential git conflict issues, as well as difficulties with code maintenance and cleanup. When working on the same base code with several developers, feature branch workflow is one of the essential techniques to use. The second, Feature branch workflow involves creating numerous branches from the master that are dedicated to various features before merging them into the master branch. The usage of feature branch workflow allows each developer to work on various features with little to no merge conflict and independent git commit logging. Similar to the git feature branch process, the third is git feature workflow with develop branch which has the additional functionality of holding the develop branch in parallel with the master branch. The development branch contains the most recent modifications made in preparation for the upcoming release. The develop branch is used to generate the developer or feature branch. The feature is tested when it is finished and then merged with the development branch. Once more, the test is run on the develop branch to check for any conflicts before being merged with the master branch. The fourth is a git-flow methodology where the Git feature workflow with develop branch is extremely similar. A release branch and a hot-fix branch are features unique to the git-flow methodology. Instead of using the develop branch, the hot-fix branch is solely made from the master branch and is then immediately merged with the master. Without having to wait for the subsequent release cycle, this branch enables immediate software deployment. The fix-branch is delivered after being merged with the master branch and with the release and development branches. This makes certain that everyone who makes a new branch has the most recent code. The develop branch is used to produce the release branch after it has all the features required for the release branch to properly merge into it. The fifth one is the git fork workflow, which is common among teams that employ open-source software. Developers build a formal repository and often copy it into their accounts in this approach. Afterward, clone that repository into your system. To make modifications and commits, developers build a new feature branch in their local repository. Pushing the modifications and the branch into the developer's repository. A pull request is issued from the branch, reviewed by the manager of the official repository, and then accepted for merging into the official repository (Khoushik, 2022).

The git feature workflow was one of them that was applied to the project. Utilizing this project was done so that the project's features may be organized. The developers are given a separate set of features for each branch. Each developer established a unique branch to code the portions of the assignment that were given to them when the task was divided among the developers. Code should be committed to the branch whenever development is complete. A pull request is later sent to the master branch. The team leader reviews the code when a pull request is submitted to the master, checks for merge conflicts, and then approves. Code conflicts were reduced thanks to this functionality, and it was simple to monitor everyone's performance. First of all, a GitHub repository was made and all team members were invited to the repository. The scrum master cloned the repository and pushed a new Flutter and NodeJS project on it using Git Bash. Then rest of the members cloned the repository and started working on their branches. Similarly, GitHub Desktop was used most of the time to create, fetch, pull, push branches, and create pull requests because GitHub Desktop allows those operations without manual coding on the terminal which made it very easy for every team member.

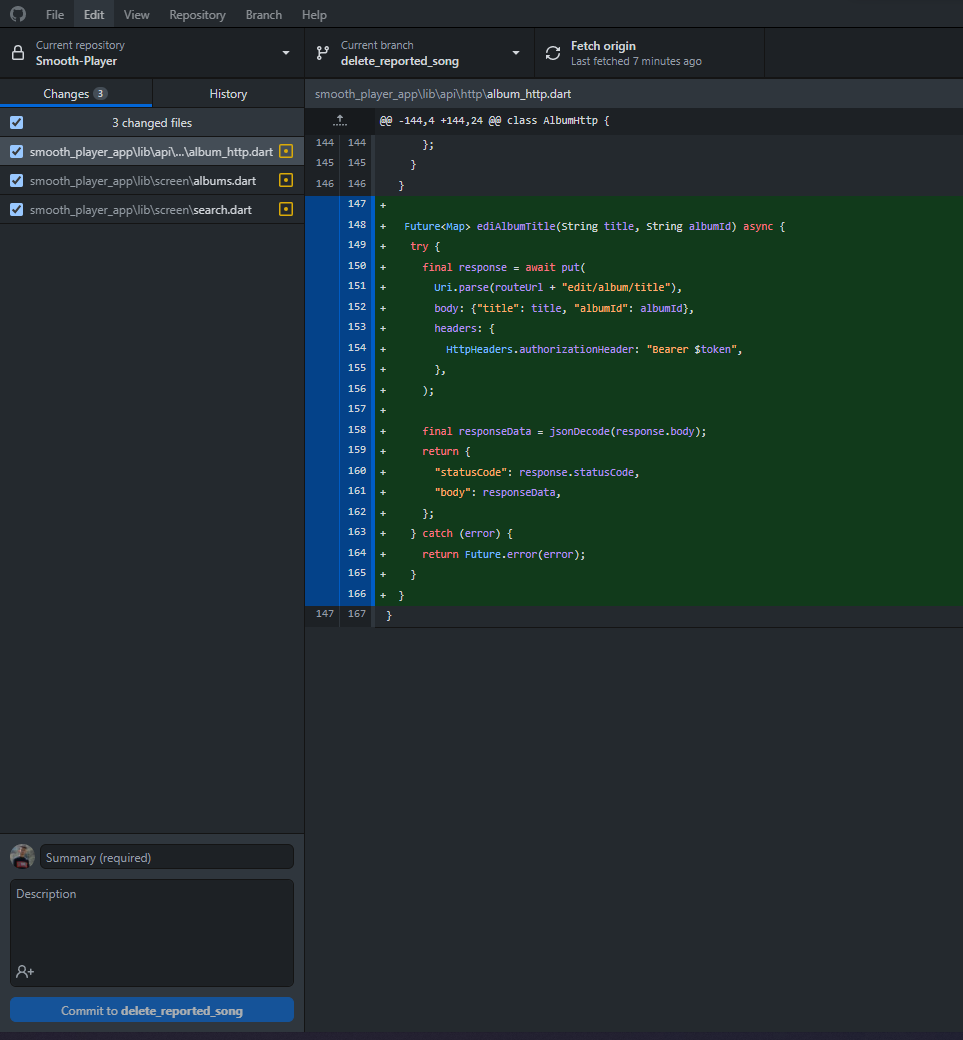


Figure 1: Working on a branch with GitHub Desktop

Git Bash was used only a few times. During each sprint, each team member created a new branch according to the user stories assigned to them and created a pull request to merge their new codes into the master branch. Team members except the scrum master were not allowed to edit and push the code on the master branch because it was the main branch and it should not contain any bugs or errors in the code. The pull requests were created after completing a feature or part of a feature so that other team members also can use the code and start working on their branch too whereas sometimes the pull request was also created at the end of the sprint. Lots of conflicts arose while merging branches on the master branch at the starting period. Then merging was done after resolving the conflicts. Every team member also started updating their branch from the master branch before merging and then no conflicts arose. On the other hand, a few times conflicts also arose while updating from the master branch. Other members could merge a branch into the master branch but required at least two approvals from the team members. Only the scrum master could merge the codes even without the approvals and most of the merging was done by him. No branches were deleted after completing the related feature. Working on multiple branches and merging were difficult during the initial sprints but after facing and resolving the problems related to it, then it was easy and no problems regarding conflicts arose. A brief description of the app and its features along with screenshots of the app's different pages were also provided on the GitHub repository.

# **Testing**

## **TDD**

TDD, or test-driven development, is simply a test-first development methodology in which test cases are written before sufficient production code, i.e., the product is not yet fully completed but the tests have been written. (Unadkat, 2021) In this development process, test cases are first created and run to fail the code. Based on these test cases, a code is then developed and implemented, and that test case is then run again. If the test is successful, a different test is run; if it is unsuccessful, changes are made to the existing code, or the code is refactored; or a new test can be created as it is a cyclical process. The basic idea behind TDD is to create and fix any failed tests before creating new code. By writing little amounts of code at a time to pass tests, we can prevent duplication of code. TDD reduces significant constraints that hinder the production and delivery of high-quality software. The system changes and improves based on user feedback, problem corrections, and the inclusion of new features to make sure everything functions as intended. TDD improves communication between the client and the development teams. Teams don't have to spend time writing elaborate test scripts because the tests have already been written. Even though TDD does not ensure accuracy, it protects our expectations of the code. There is no questioning the enormous usefulness of unit tests, yet the concept of accuracy through TDD is founded on expectations rather than reality. (qavbox, 2021)

For the TDD testing part, schema testing of the mongoose models was done. Inserting, deleting, and updating crud operations were done in the schema testing. The tests were done using the “jest” npm library. During the schema test, some of the tests failed because of the expected wrong data instead of the inserted data. The test failed because the inserted “id” would not exist in the database while testing the update operation and also expecting only String for object id instead of converting the String into object id and then testing. The tests were done by all of the team members and the object id used on the test was only available in the database of the team member who did that test. This also caused problems because those tests failed on another team member’s project. Therefore, the ids were changed by the scrum master after all sprints were finished and testing of all models was done and all tests passed at last. Some screenshots of tests are shown below.

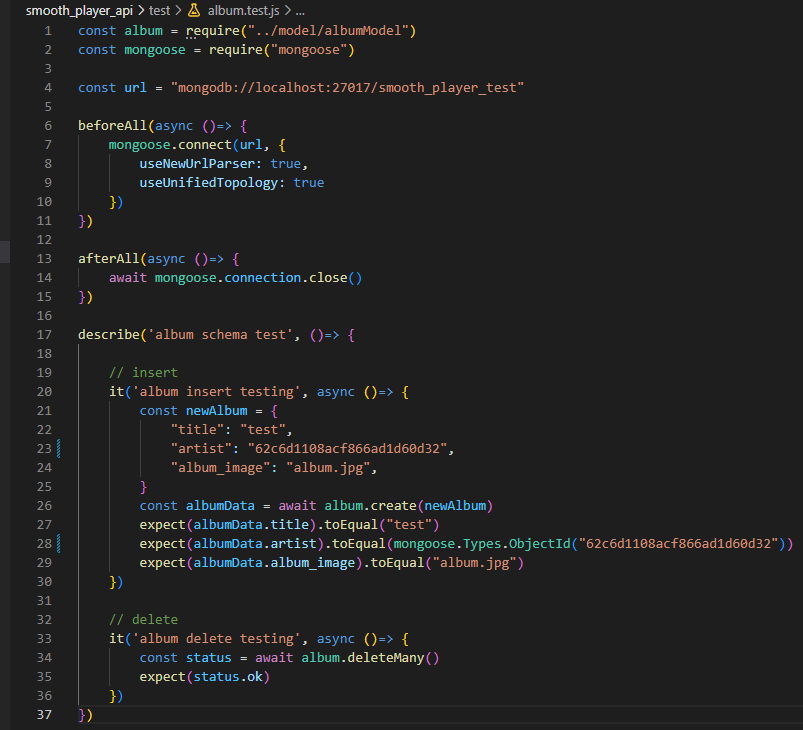


Figure 2: Album model insert and delete test



Figure 3: Featured playlist model insert and update test

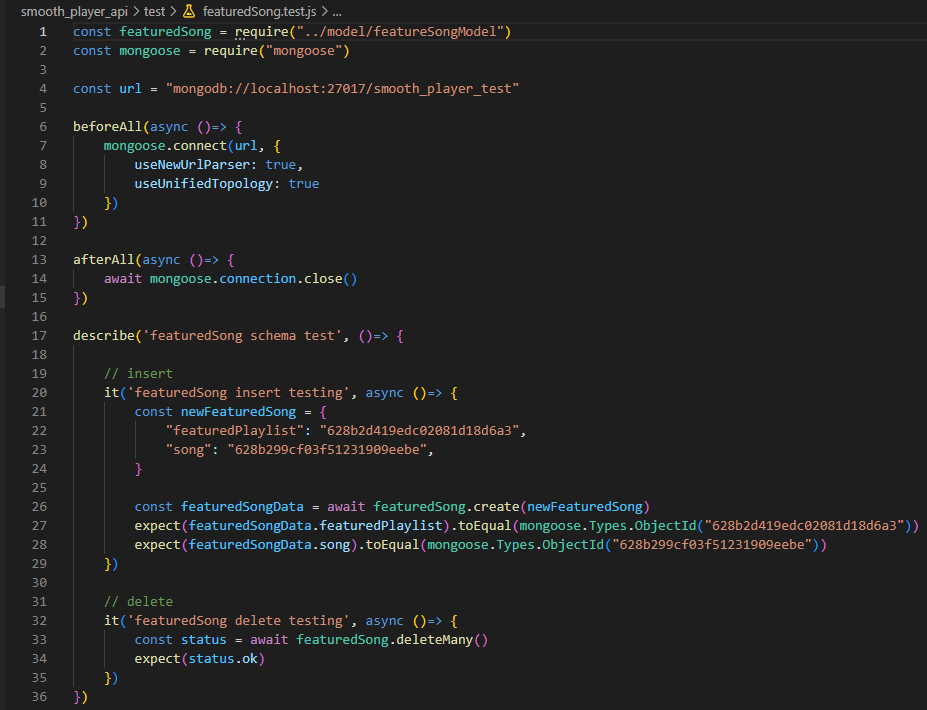


Figure 4: Featured Playlist Song insert and delete test



Figure 5: Like model insert and delete test



Figure 6: Report insert and delete test

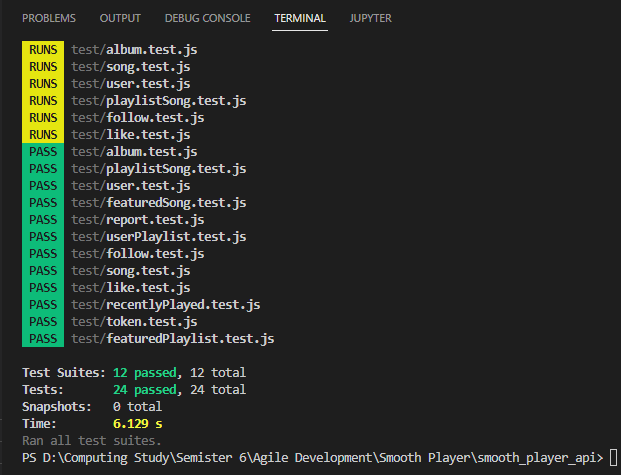


Figure 7: All TDD tests completed successfully

## **BDD**

BDD, or behavior-driven development, is all about team collaboration, specifically between the business and technical teams in any organization. Rather than writing test cases or test scenarios, behavior-driven development instead creates behaviors for applications that address both functional and non-functional requirements. (Stefanski, n.d.) BDD isn't a completely fresh take on TDD for software development. BDD is only a layer on top of TDD, not a full avoidance of TDD by developers. As the system develops, tests that assert certain states in TDD may provide incorrect results. However, BDD focuses on behavior, such as what would happen to this system under certain circumstances. The main goal of BDD is to close the knowledge gap between technical and business teams. In an organization, a typical BDD process begins with a user story in an agile approach, followed by a discussion on how the system should act. Following the discussion is the coding phase, when the implementation is carried out by the documentation. An automation test is built during the coding process, but it will fail since no code has been produced. Ultimately, the code will be written to make a particular test successful. The most significant benefit of BDD is the increase in code quality, which lowers development business risk associated with different expenditures like management.

Similarly, for the BDD testing part, widget testing of the flutter stateless and stateful classes was done. Flutter default testing package was used to test the classes. A single class was tested in a single test file and widgets like Text, ElevatedButton, Icon, Form, Column, SingleChildScrollView, and TextFormField were tested. During the testing period, the HTTP client errors were faced most of the time and most of the testing failed because of this. The widgets which contain Http classes or the classes containing HTTP classes in the initState function failed. Therefore, only those widgets and classes which does not contain HTTP client errors were tested successfully. Most of the tested classes contain TextFormField and ElevatedButton. The tests were done by all of the team members and some already passed tests failed at last. This was because some modifications were done on those tested classes until the last sprint. Therefore, the failed tests were also modified, and then the tests passed at last. Some screenshots of the tests are shown below.



Figure 8: Edit Song class test

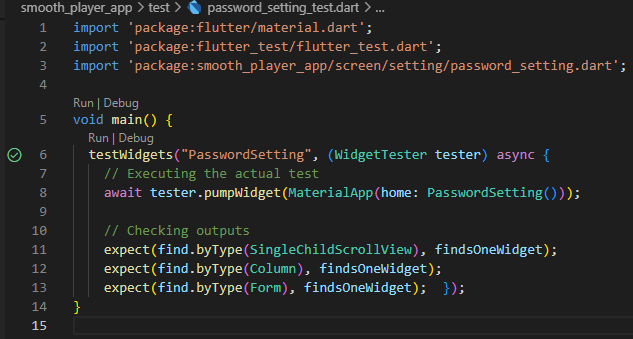


Figure 9: Password Setting class test



Figure 10: Sign Up class test

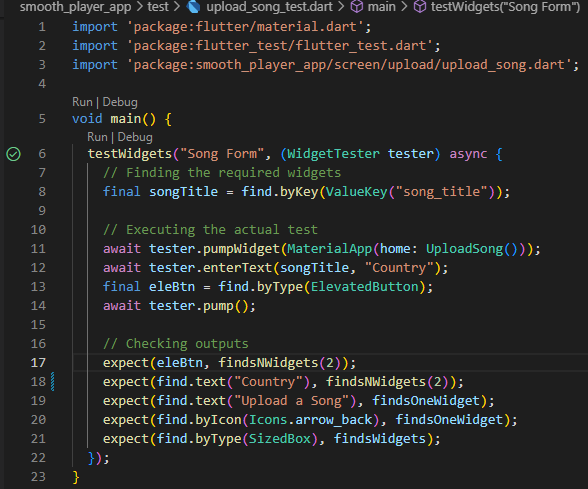


Figure 11: Upload Song class test

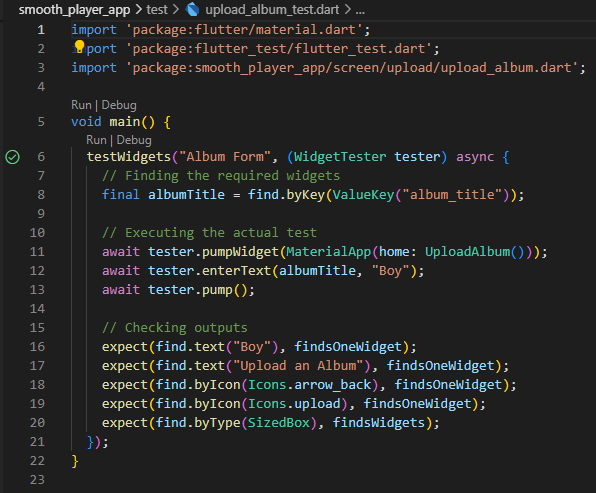


Figure 12: Upload Album class test

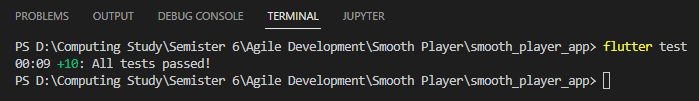


Figure 13: All BDD tests completed successfully

# **Conclusion**

Working on git during this project has enhanced every team member’s version control skill. Git feature workflow was used to complete all the user stories in every sprint. It was difficult working on multiple branches because of the conflicts while merging the codes into the master branch at the starting phases but every team learned to avoid the conflicts. Merging was usually done at the end of the sprints and sometime before it. Testing was also the major part of this project and both TDD and BDD testing were done. Multiple Mongoose models were tested in TDD testing and multiple Flutter widgets were tested in the BDD testing. The tests were done by each of the team members and some problems also occurred during the testing and they were solved too. However, Testing reduced the bugs of the application and helped to fix the errors as fast as possible.

# **Links**

**GitHub:** <https://github.com/gaurishankar007/Smooth-Player.git>

## **YouTube:** <https://youtu.be/JvnJoakBgYA>

# **References**

Khoushik, V., 2022. *RovitPM.* [Online]   
Available at: https://rovitpm.com/5-git-workflows-to-improve-development/  
[Accessed 7 June 2022].

qavbox, 2021. *YouTube.* [Online]   
Available at: https://www.youtube.com/watch?v=FmhYMjO2cVg&t=675s  
[Accessed 22 07 2022].

Stefanski, R., n.d. *AltexSoft.* [Online]   
Available at: https://www.altexsoft.com/blog/behavior-driven-development/  
[Accessed 12 07 2022].

Unadkat, J., 2021. *BrowserStack.* [Online]   
Available at: https://www.browserstack.com/guide/what-is-test-driven-development  
[Accessed 13 07 2022].