System Test Scenarios:

Sprint 2:

User Story 1: As a user, I want to listen to new instrumental music, so that I don't hear the same song twice.

Scenario:

1. Visit site.
2. Scroll to the second page or click “Player” in the navbar; user should be directed to the “Player” page.
3. User should see a play button; click play and music begins to play.
4. Once a song ends, a new song should automatically start to play after it based on the randomization algorithm.

User Story 2: As a user, I want that website to have a minimalist design so that users can access the features easily

Scenario:

1. Visit site.
2. Scroll through pages to see smooth transitions and minimal design/artwork.
3. User should see a fixed navbar at the top of the page always.
4. User should easily be able to navigate between pages by a single click on the links in the navbar.
5. All features should be able to be interacted with a single click.

Sprint 3:

User Story 1: As a user, I want to save music that I listened to so that I can listen to the music again if I liked it.

Scenario:

1. Visit site.
2. Scroll to the second page or click “Player” in the navbar.
3. User should see a save button to the right of the play button.
4. Click on the save button; user should receive a local download of the mp3 file.

User Story 2: As a user, I want to share a track with someone else so that they may enjoy the music too.

Scenario:

1. Visit site.
2. Anywhere on the website, the user should see social icons for twitter and facebook as well as a clipboard icon in the navbar.
3. Click on the facebook or twitter icon; the user should be redirected to facebook or twitter specifically to share the website.
4. Click on the clipboard icon; the user should see an alert notifying them that the website link has been copied to their clipboard.

User Story 3: As a user, I want a parallax design, so that I can better immerse myself in the style of the website.

Scenario:

1. Visit site.
2. Scroll anywhere on the website, the elements, which are on different layers, should move up and down slightly at different speeds and overlap with each other depending on their layer position.

User Story 4: As a user, I want seamless audio controls so that the controls do not disrupt my listening immersion.

Scenario:

1. Visit lofiai.herokuapp.com.
2. On the “Player” page while music is playing, click on the pause button; the music currently playing should be stopped immediately and a play button should appear in place of the pause button.
3. Click on the play button; the music should start playing again and the pause button should reappear in place of the play button.
4. Click on the volume slider and slide it to the left; the music should decrease in volume until muted.
5. Click on the volume slider and slide it to the right; the music should increase in volume until maxed out.
6. User should experience no disruptions in their experience visually or in terms of delays between clicking buttons and resulting effects.

Sprint 4:

User Story 1: As a user, I want to read about the product so that I know what the product is for.

Scenario:
1. Visit the site.
2. Scroll to the third page or click “About” in the navbar; user should be directed to the “About” page.
3. User should see a brief description of the project on the left side and a list of current project contributors.

User Story 2: As a user, I want the website to be hosted so that I can access the website.

Scenario:
1. Visit lofiai.herokuapp.com. The website is hosted on Heroku and available for the public to see.

User Story 3: As a user, I want to read about the AI's progress so that I can fully understand what went into this project.

Scenario:
1. Visit the site.
2. Navigate to the “Learning with LoFi” section.
3. Click on the different topic tabs to read about the project’s documentation.

User Story 4: As a user, I want to be able to read the track name and know how far into the track I am, so that I can refer to them

Scenario:
1. Visit the site.
2. On any page of the website, the user should see “You’re listening to track #” and then the track number, track name and current track progress in the navbar.

Unit test

Playing music:

- Regarding the back-end, playing music required setup of a database and routes. Database setup was handled in server.js, which connected to a MongoDB database used to store links to mp3 files stored on AWS. Routing specifically for retrieval and storing of songs was done in routes/songs.js.
- Regarding the front-end, playing music required axios to connect to the back-end and retrieve the song path in Song.js.
- Equivalence Classes:
    - In server.js, mongoose successfully connects to the database, given the link.
    - On "/get/song" route, we get a list of every "song" entry in the database.
    - In Song.js, axios successfully retrieves list of all "song" entries using the "/get/song" route and stores this data in this.state.data.
    - On clicking play, the audio element, given a song path from the state data's list, will successfully play the song using the path.
- Tests:
    - To check if Mongoose successfully connected to the database, we logged in the console whether mongoose.connect(link) was successful or not.
        - If successful, the console logged "Start".
        - Otherwise, the console logged "App starting error:" along with the error stack.
    - To test the "/get/song" route, we tracked the response of the get method through logs to console.
        - If successful, the console logged an array of every song entry in the database.
        - Otherwise, we analyzed the output to see what was missing or incorrect.
    - To test whether axios successfully retrieved database information in the front-end, after supplying axios with the "/get/song" route, we logged the response in console

to see if it matched what was in the database. Then, we stored this response data in this.state.data.
    - To test actual playing of music and whether the song paths are valid, we supplied an audio element with any song path in this.state.data and checked to see whether music actually played.

Learning With Lofi Page:

- The “Learning With Lofi” page is a React component titled Learn.js that is called in App.js.The goal of which is to have a collapsible folder like component that users will be able to interact with by clicking topic titles in the component and the component will respond by revealing the text written for said topic.
- Equivalence Classes:
    - In Learn.js we create a radio HTML type element that is styled to display collapsible content.
    - In divs of class titled "accordion-title” we’ll have the title of the content we’re portraying.
    - In the divs titled “accordion-content” we’ll have the content we’ve written that helps explain elements of our product LoFiAi.
- Tests:
    - To check if the “Learning With Lofi” successfully displays the written content we click a title of the topic the user wants to read.
        - If successful, the content related to the title will slide out to display to user.
        - Otherwise,nothing will happen.
    - What users should first see when they’re on the Learning with Lofi collapsible element page.
        - If rendered successfully, “What is this page for?” content will appear.
        - Otherwise, another topic will display. This would be a failure in the component.
    - To test whether a title highlights when hovering a mouse cursor over the title.
        - If it highlights green, then it has rendered successfully.
        - Otherwise, it did not highlight which means we have a failure.
    - To test if the content title remains highlighted green when on the selected content.

        - If successful, a user clicks on a panel it will stay highlighted green as you read the content.
        - Otherwise, It will not highlight which will be a failure in component.

NavBar:

- The navbar follows the page and sticks towards the top when scrolling
- Equivalence Classes:
    - Links
        - Link to current page
        - Link to a different page
    - Current Page Links
        - Home
        - Player
        - About
        - Learning With LoFi
- Tests:
    - To test the links, we scroll to each individual page and click a link to that same page, and a link to a different page, to make sure the link brings us to the right page, regardless of the current page it is on
        - The Home link should bring the user to the Home page
        - The Player link should bring the user to the Player page
        - The About link should bring the user to the About Me page
        - The Learning With LoFi link should bring the user to the AI Documentation page
        - If all cases work regardless of page, the test has completed

Sharing:

- Sharing has two links that shares the current song with either Facebook or Twitter. It uses the current song variable in Song.js.
- Equivalence Classes:
    - Songs
        - Currently playing song
    - Links
        - Link to Twitter
        - Link to Facebook
- Tests:

    - To test the sharing with social media, we either click the Twitter icon or the Facebook icon.
        - The Twitter icon opens a new tab to Twitter with the current song's address copied to a Twitter tweet. The Facebook icon opens a new tab to Facebook with the current song's address copied to a new Facebook post.
        - If neither of these happen, then nothing has happened and the test fails.

Timer:

- The timer uses data collected in App.js via callback functions called in Song.js to report and update the song name, as well as the current time elapsed in the song, and the length of the song.
- Equivalence Classes:
    - Songs
        - First song loaded
        - All subsequent songs (can be tested via the second song to play)
    - Time in a given song:
        - Song currently loading
        - Song about to play
        - Song currently playing
        - Song currently paused
        - Song unpaused
- Tests:
    - We ensured the data reported was accurate by pressing "play" and observing what was written during:
        - The first song loading
            - This reports "" as the title and 0/0 as the time
        - The first song about to play
            - This reports the title of the song, followed by 0/(the song's duration)
        - The first song playing
            - This reports the title followed by (current seconds into the song)/(duration of the song)
        - The first song paused
            - This reports the same as the above case, with the time no longer increasing each second
        - The first song unpaused
            - This reports the same as the first song playing normally

- The second song was then tested and reported back in the exact same way as for the first song in each case