TEAM 4

BY HASAN , MICHAEL , TOM AND JORDAN

Brief

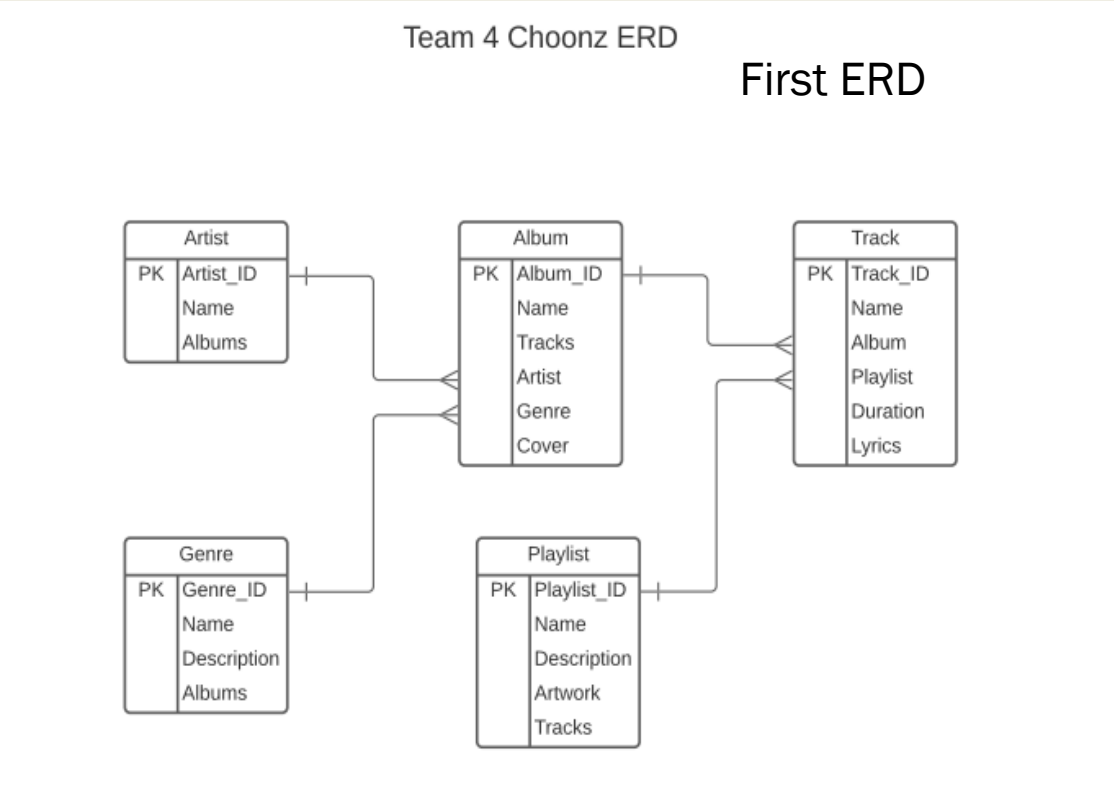
The aim of this project was to Create an Implement a Music Hosting Web application using HTML, CSS, Bootstrapp and Javascript for the construction of the Front end While the back end utilised Java and the SpringBoot framework where the program will be rigorously tested throughout the Software Development Life Cycle

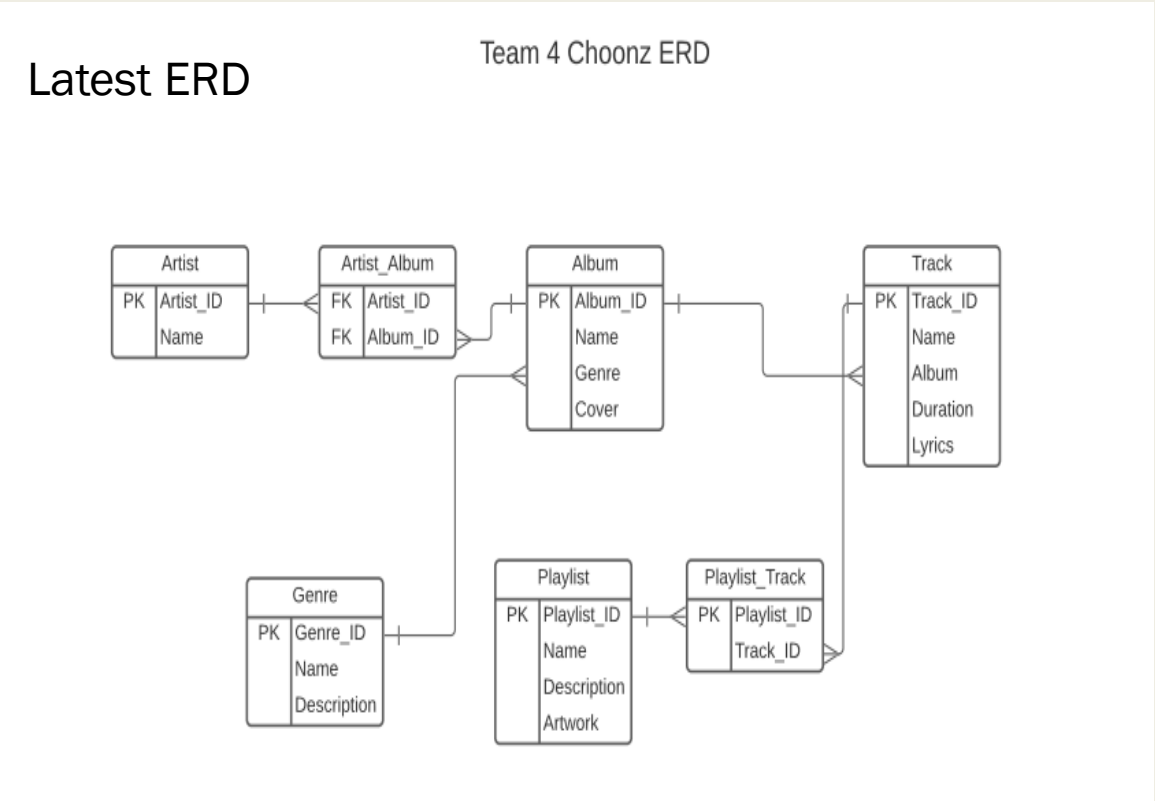
Planning

In order to assess what the requirements of the project will be, we must create the following

- Entity relationship diagram
- Universal Modeling Language
- Risk Matrix
- User stories

ERD


Team 4 Choonz ERD
First ERD
Artist
PK Artist_ID
Name
Albums
Album
PK Album_ID
Name
Tracks
Artist
Genre
Cover
Track
PK Track_ID
Name
Album
Playlist
Duration
Lyrics
Genre
PK Genre_ID
Name
Description
Albums
Playlist
PK Playlist_ID
Name
Description
Artwork
Tracks

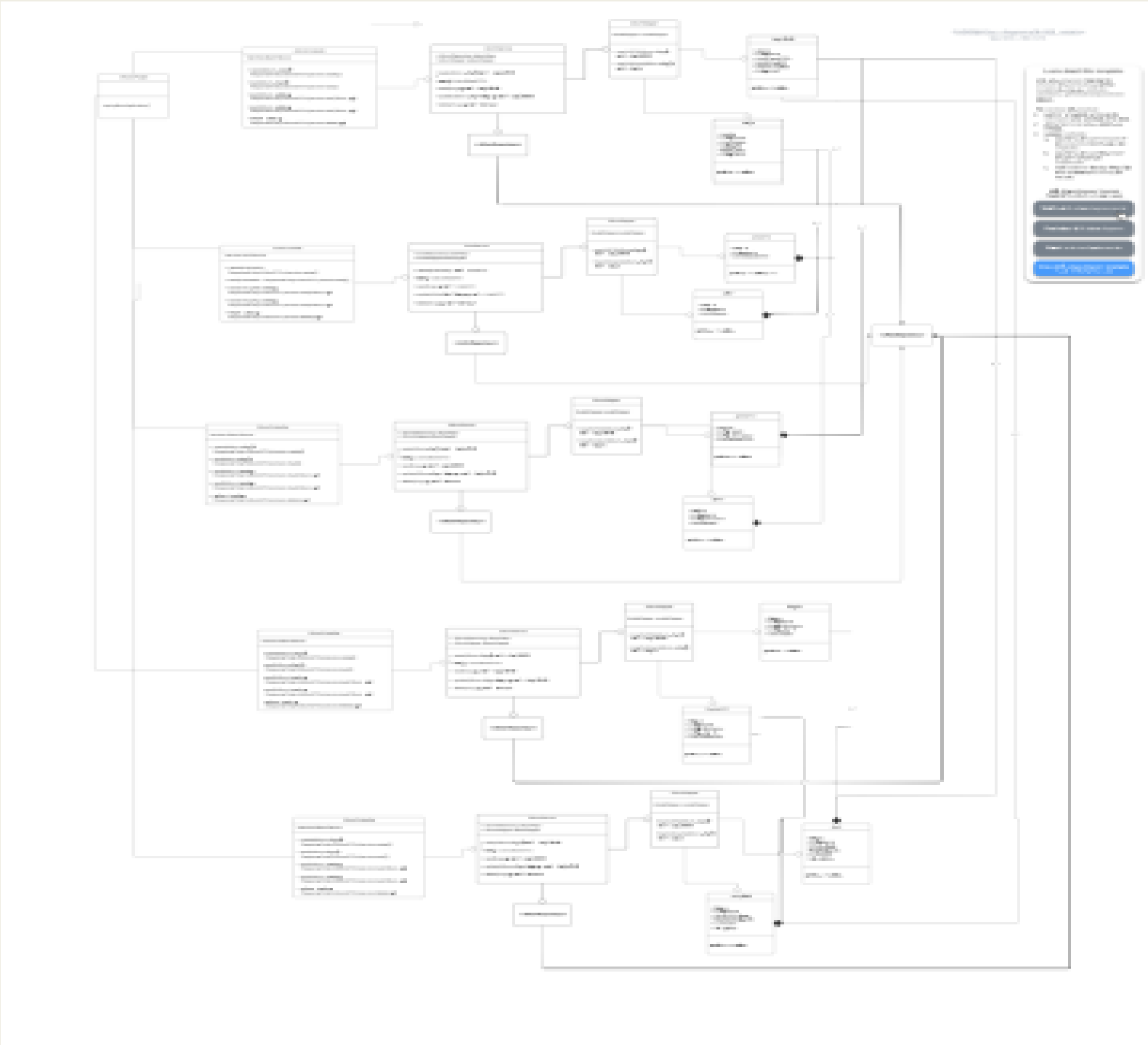

Latest ERD
Team 4 Choonz ERD
Artist
PK Artist_ID
Name
Artist_Album
FK Artist_ID
FK Album_ID
Album
PK Album_ID
Name
Genre
Cover
Track
PK Track_ID
Name
Album
Duration
Lyrics
Genre
PK Genre_ID
Name
Description
Playlist
PK Playlist_ID
Name
Description
Artwork
Playlist_Track
PK Playlist_ID
Track_ID

Risk Assessment

		Consequence				
		Negligible 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Likelihood	Rare 1	Low 1	Low 2	Low 3	Moderate 4	Moderate5
	Unlikely 2	Low 2	Moderate 4	Moderate 6	High 8	High 10
	Possible 3	Low 3	Moderate 6	High 9	High 12	Extreme 15
	Likely 4	Moderate 4	High 8	High 12	Extreme 16	Extreme 20
	Almost Certain 5	Moderate 5	High 10	Extreme 15	Extreme 20	Extreme 25

Risk	Statement	Response	Likelihood	Consequence	Risk
Internet outage	Internet may cut out	Fix it or wait for it to be fixed	3	1	3
Computer Damage	Computer could break	Fix it or get a new one	1	2	2
Using new software	Using unfamiliar difficulties will bring delays	Spend time learning new software	5	2	10
Poor health	Unforseen health issues	Response unique to each situation	1	4	4
Additional requirements	More requirements could be added to the project	Adjust agile methods	2	1	2
Time required elsewhere	I may be required to attend training days	Adjust agile methods	5	1	5
Unclear requirements	Initial requirements unclear	Alter goals as project progresses	3	2	6
Help unavailable	Trainers may be too busy to help with issues	Wait or continue trying solutions	4	1	4
Low Motivation	Low motivatino reduces productivity	Schedule work effectively	3	3	9
Data corruption	Data could be lost or become corrupt	Use a recent git backup	1	1	1
Merge Conflicts	Merge conflicts could cause issues	Effective communication	3	2	6
Communication Failure	Poor communicatioin cause more issues	Effective communication	1	3	3
Group conflict	People could fall out	Pull your weight and be friendly	2	3	6

UML



Planning – Using Jira

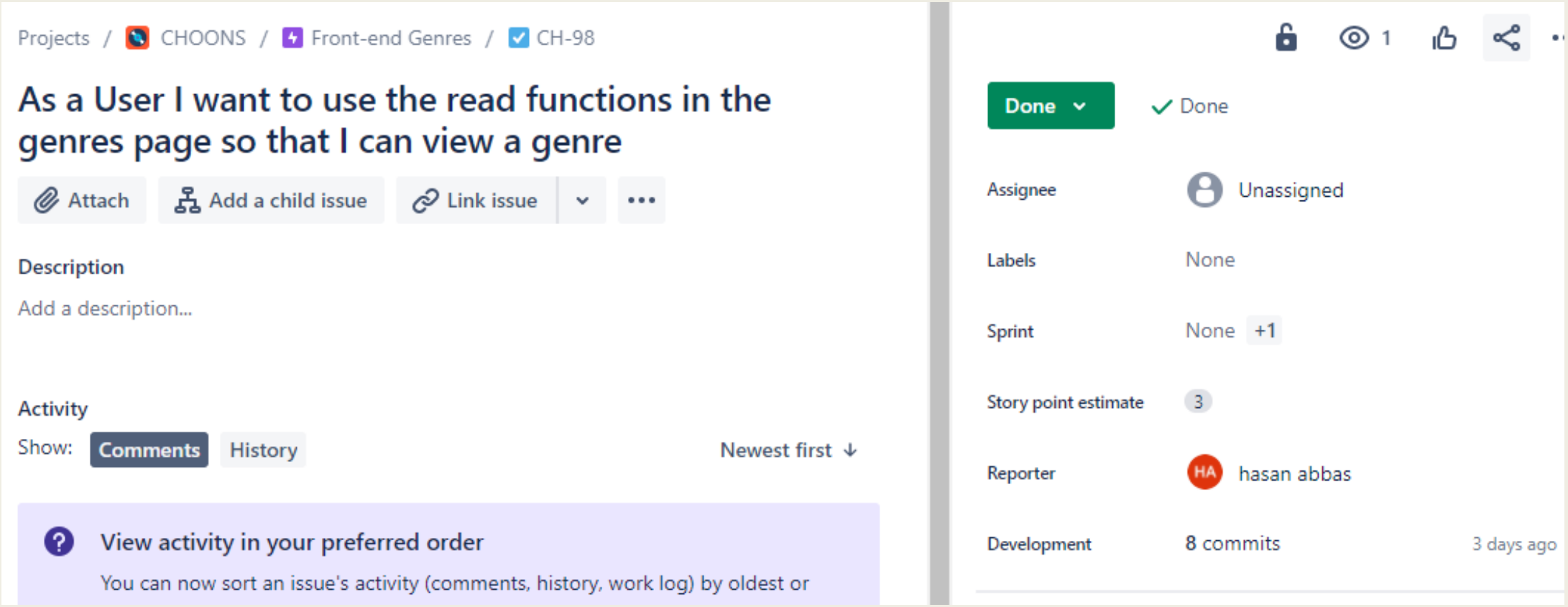
Jira is a powerful web-based work management that allows us to plan, manage and track the workflow of the project whilst also allowing us to create and track epics, sprints and tasks.

Planning – Using Jira – user stories

User Stories are a highly effective way of determining the software requirements based on what the user would typically expect the software to do. For example,
"As a User I want to use the read functions in the genres page so that I can view a genre"

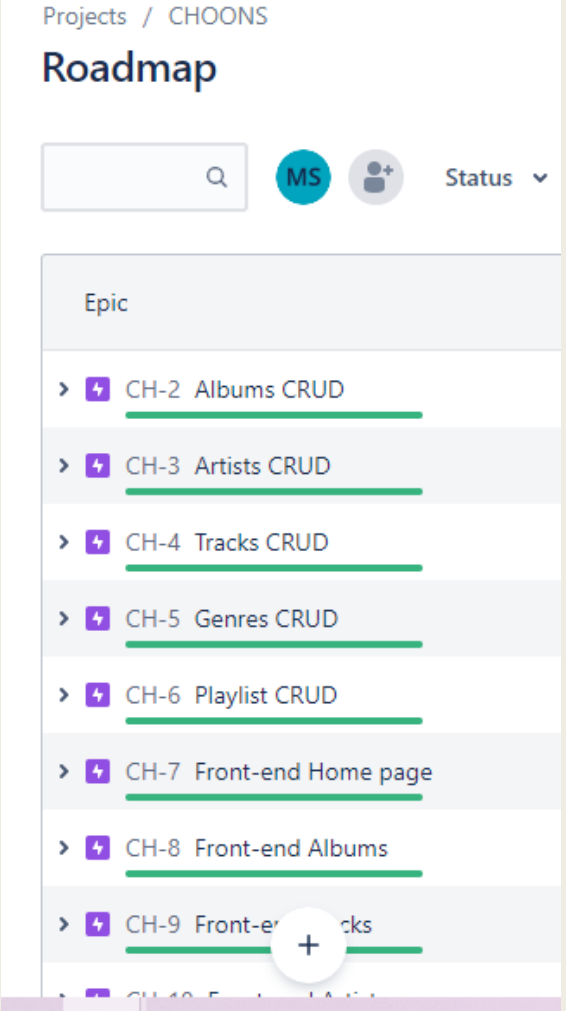
However, with jira, tracking the progress of a task becomes a lot more streamlined with the use of smart commits

Planning – Using Jira – user stories issue example



Planning – Using Jira -creating epics

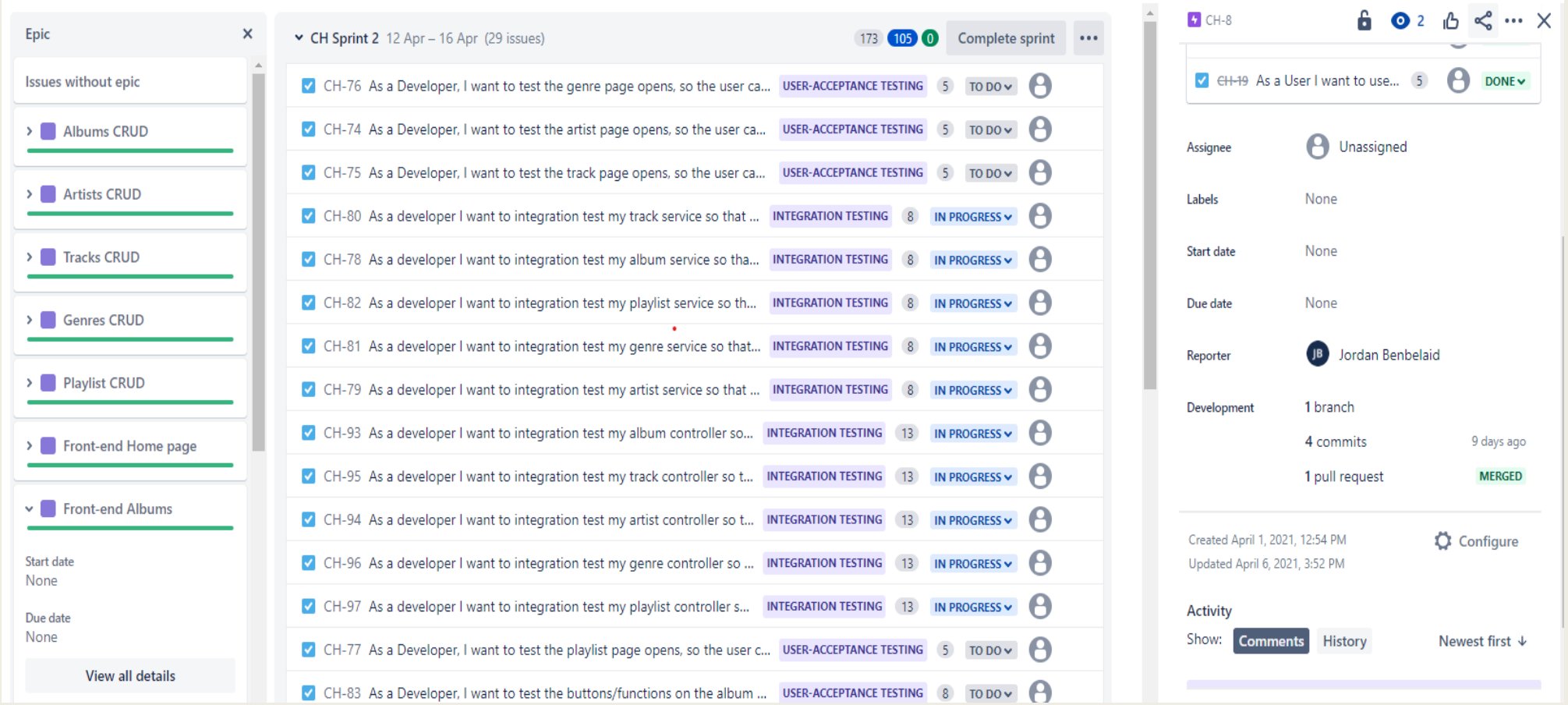
Epics are essentially a large body of work that can be broken down Into smaller tasks or user stories(issues in Jira)



Planning – Using Jira – planning sprints

Sprints are typically a set period of time in which a specific work must be completed and ready for review. Each sprint usually consists of a planning meeting with the product owner and the development team to agree on what exactly needs to be accomplished by the end of the sprint. Jira however, simplifies the process of planning sprints with the ability to not only plan what task needs to be completed when but as we seen before with smart commits, it also allows us to track who did what.

Planning – Using Jira – planning sprints



Planning – Daily Stand-Ups

During each sprint period, there was a daily standup. The purpose of the daily stand up was to discuss what each group member worked on, what they plan to work on next and what isses they had with any given task (known as blockers).

Planning – Daily Stand-Ups

07/04/21

Hasan

- Home page + album page + artist page. Wire frame
- Finish off look for front end
- No blockers

Tom

- Service integration tests and a little on unit tests
- Controller integration tests
- No blockers

Jordan

- Home page + album page + css.
- Finish off front end, aid the back end
- No blockers

Michael

- Started unit tested controller classes
- Service unit tests – remaining controller unit tests
- Some errors with playlist tests

15/04/21

Hasan

- Troubleshooting Selenium - finally got it working with cucumber
- Finish up JavaScript quickly then carry on with selenium
- No blockers

Tom

- Fixed join table and controller integration tests
- Finish fixing CI tests and implement collab artists filter
- No Blockers

Jordan

- Jmeter testing, getting HTML and .csv reports sorted
- Continuing with Jmeter testing – compiling reports
- Time required to run each test

Michael

- Troubleshooting Selenium with cucumber, succeeded eventually
- Carry on with selenium – testing user inputs
- No Blockers

Initial Functionality of the program

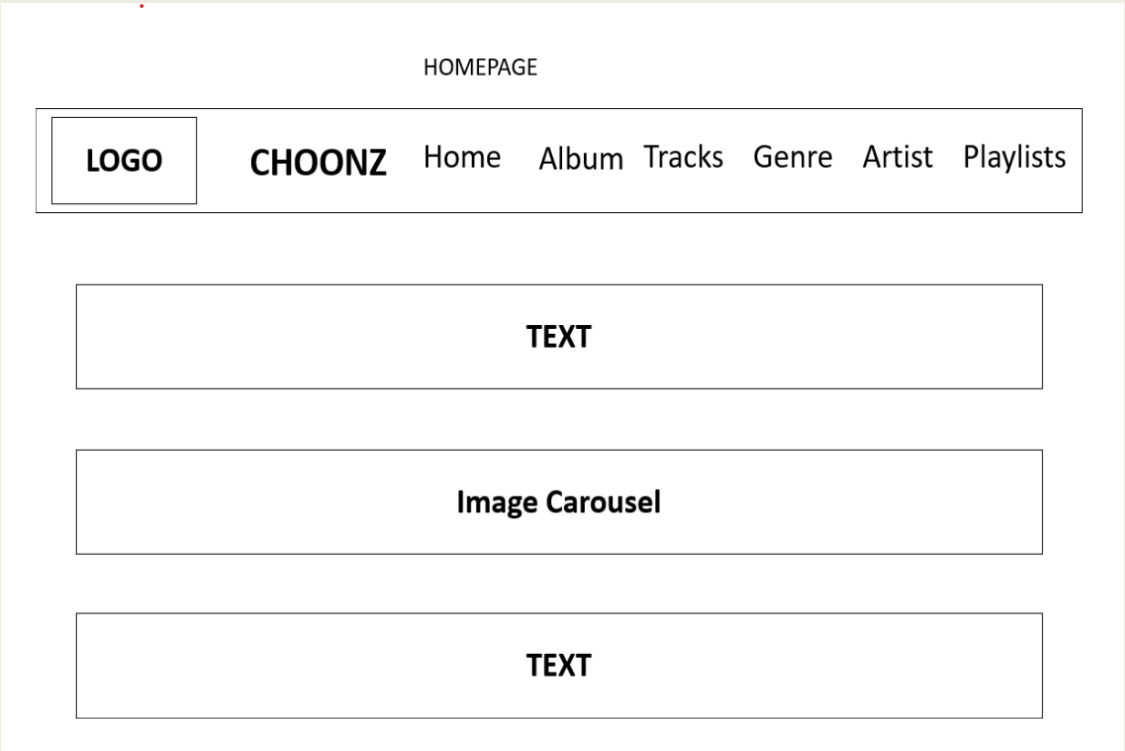
Segment	Method	Success			Issues:
Album	Post	Yes			Put mappings are post mappings
Album	Get	Yes			Creating genres and tracks doesn't work
Album	Put	No			
Album	Delete	Yes			
Artist	Post	Yes			
Artist	Get	Yes			
Artist	Put	No			
Artist	Delete	Yes			
Genre	Post	No			
Genre	Get	Yes			
Genre	Put	-			
Genre	Delete	-			
Playlist	Post	Yes			
Playlist	Get	Yes			
Playlist	Put	No			
Playlist	Delete	Yes			
Track	Post	No			
Track	Get	Yes			
Track	Put	-			
Track	Delete	-			

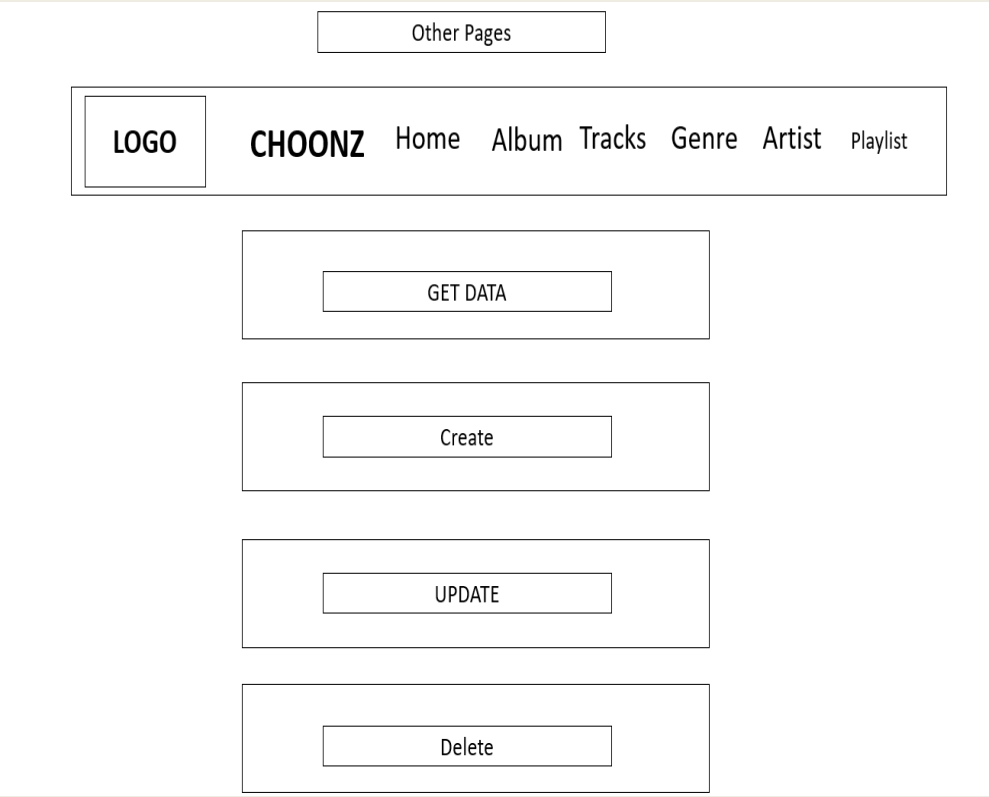
The Front End

The front end was programmed in HTML, CSS, Bootstrap and javascript. However, in order to ensure the front end would operate as a given user would expect it to, it was rigirously tested using Selenium, Cucumber and Gerkhin.

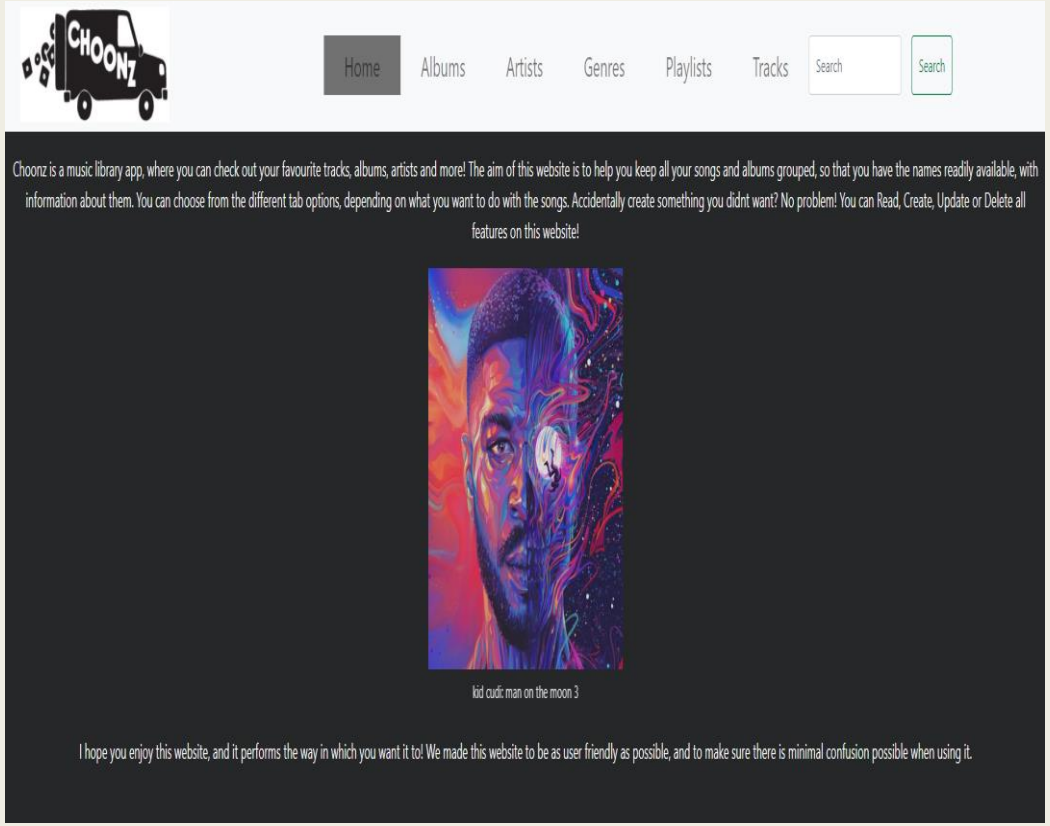
However, In order to plan the layout of the web application, Wire-Frame's where used to create a template for the web application.

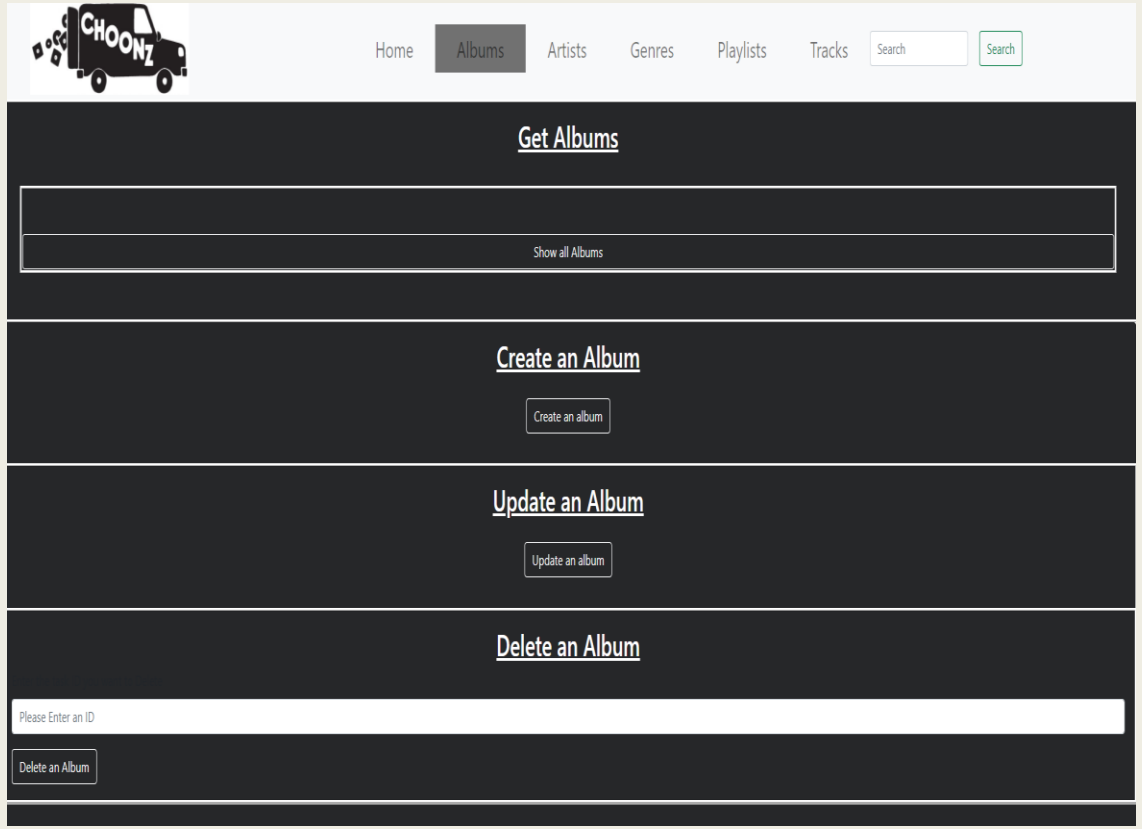
The Front End - Wire-Frame's





The Front End – Home Page





The Front End – Javascript

```javascript
function createAlbumTable(data){

  console.log(data);
  deleteTable();

  let table = document.getElementById("AlbumResultSet");
  let row =  document.createElement('tr');
  let idHeader = document.createElement('th');
  let nameHeader = document.createElement('th');
  let TrackHeader = document.createElement('th');
  let coverHeader = document.createElement('th');
  let artistHeader = document.createElement('th');
  let genreHeader = document.createElement('th');
  idHeader.innerHTML= "Album-ID";
  nameHeader.innerHTML = "Album Name";
  TrackHeader.innerHTML = "Track Name";
  coverHeader.innerHTML = "Album Description";
  artistHeader.innerHTML = "Artist ";
  genreHeader.innerHTML = "genre ";
  row.appendChild(idHeader);
  row.appendChild(nameHeader);
  row.appendChild(TrackHeader);
  row.appendChild(coverHeader);
  row.appendChild(artistHeader);
  row.appendChild(genreHeader);
  table.appendChild(row);

  for(let i = 0; i<data.length; i++){
    console.log(data[i].name);
    let row = document.createElement('tr');
    let cell1 = document.createElement('td')
    let cell2 = document.createElement('td');
    let cell3 = document.createElement('td');
    let cell4 = document.createElement('td');
    let cell5 = document.createElement('td');
    let cell6 = document.createElement('td');

    cell1.innerHTML = data[i].id;
    cell2.innerHTML = data[i].name;
```

```javascript
  for(let i = 0; i<data.length; i++){
    console.log(data[i].name);
    let row = document.createElement('tr');
    let cell1 = document.createElement('td')
    let cell2 = document.createElement('td');
    let cell3 = document.createElement('td');
    let cell4 = document.createElement('td');
    let cell5 = document.createElement('td');
    let cell6 = document.createElement('td');

    cell1.innerHTML = data[i].id;
    cell2.innerHTML = data[i].name;
  //  cell3.innerHTML = data[i].tracks[i].name;
  for(let j=0;j<data[i].tracks.length;j++){
    let row1 = document.createElement('tr');
    let innercell = document.createElement('td');
    innercell.innerHTML = data[i].tracks[j].name;
    row1.appendChild(innercell);
    cell3.appendChild(row1);
  }
    cell4.innerHTML = data[i].cover;
    cell5.innerHTML = data[i].artist;
    cell6.innerHTML = data[i].genre;
    row.appendChild(cell1);
    row.appendChild(cell2);
    row.appendChild(cell3);
    row.appendChild(cell4);
    row.appendChild(cell5);
    row.appendChild(cell6);
    table.appendChild(row);
  }
}
```

```javascript
function createAlbum() {
    fetch("http://localhost:8090/albums/create", {
      method: 'post',
       headers: {
         "Content-type": "application/json"
      },
       body: JSON.stringify({
          name: document.querySelector("#createAlbumName").value,
          artist: {id: parseInt(document.querySelector("#ArtistName").value)},
          genre: {id: parseInt(document.querySelector("#GenreName").value)},
          cover: document.querySelector("#createCover").value

        })
       })
       .then(res => {
         if(res.status!=201){
             console.error(res)
         }
           res.json()})
       .then((data)=> {
          console.log(`Request succeeded with JSON response ${data}`);
      })
      .catch((error)=> {
          console.log(`Request failed ${error}`);
    });
    }


    function updatealbum(){
     let Aid= parseInt(document.querySelector("#updateAlbumId").value)
    fetch("http://localhost:8090/albums/update/"+Aid, {
      method: 'put',
       headers: {
         "Content-type": "application/json"
      },
       body: JSON.stringify({
          name: document.querySelector("#updateAlbumName").value,
          ArtistName: document.querySelector("#uArtistName").value,
          GenreName: document.querySelector("#uGenreName").value,
          cover: document.querySelector("#updateCover").value



        })
       })
```

The Front End – Automation Testing

As Discussed earlier, the front end was tested Using Selenium, gherkin and Cucumber where Selenium allows for the program to mimic and automate a users actions and input while Cucumber and gherkin allows for the tester to write and structure the code in a way that almost anyone can interpret

The Front End – Automation Testing - Cucumber Features

```
Feature: NavigateOnChoonz

Background:
Given I can access Choonz

Scenario: Navigating using the header
# artists page
    When user clicks on the artists tab

    Then user accecesses artists
#albums page
    When user clicks on albums tab

    Then user accecesses albums

#Playlists page
    When User clicks on playlists page

    Then user accesses playlists

#tracks page
    When user clicsk on tracks page

    Then user accesses tracks

#Genres page
    When user clicks on genres page

    Then user accesses genres page
```

Automation testing - navigation

```
@Given("^I can access Choonz$")
public void I_can_access_Choonz() {
    driver.get(Url.HOMEPAGE);

    assertEquals("Home", driver.getTitle());

}

@When("^user clicks on the artists tab$")
public void user_clicks_on_the_artists_tag() throws Throwable {
    WebElement input = driver.findElement(By.cssSelector("#ArtistsLink"));
    Actions action = new Actions(driver);
    action.moveToElement(input);
    action.click().perform();
    assertEquals("artist", driver.getTitle());
}

@Then("^user accecesses artists$")
public void Web_page_navigates_to_the_artists_page() throws Throwable{
    driver.get(Url.ARTISTS);
    assertEquals("artist", driver.getTitle());
}
```

Automation Testing – data handling

```java
@Given("^user goes to artists page$")
public void user_goes_to_artists_page() {
    driver.get(Url.ARTISTS);

    assertEquals("artist", driver.getTitle());
}

@When("^user clicks on create Artist$")
public void user_clicks_on_create_Artists() throws Throwable{

    WebElement input = driver.findElement(By.cssSelector("#createButton"));
    Actions action = new Actions(driver);
    action.moveToElement(input);
    action.click().perform();
    input.findElement(By.cssSelector("#createArtistName"));
    action.moveToElement(input);
    input.sendKeys("Rick Astley");
    input.submit();
    input.findElement(By.cssSelector("create"));

}
```

```java
@When("^user updates artist$")
public void user_updates_artist() throws Throwable{
    WebElement input = driver.findElement(By.cssSelector("#updateButton"));
    Actions action = new Actions(driver);
    action.moveToElement(input);
    action.click().perform();
    input.findElement(By.cssSelector("#updateArtistId"));
    action.moveToElement(input);
    action.click().perform();
    input.sendKeys("1");
    input.submit();
    input.findElement(By.cssSelector("#updateArtistName"));
    action.moveToElement(input);
    action.click().perform();
    input.sendKeys("tina turner");
    input.submit();
    input.findElement(By.cssSelector("div.modal-footer:nth-child(3) > button:nth-child(1)"));
    action.moveToElement(input);
    action.click().perform();
}


@Then("^user clicks read artists again$")
public void user_clicks_read_artists_again() throws Throwable{
    read("tina turner");
}

@When("^user deletes artist$")
public void user_deletes_artist() throws Throwable{
    WebElement input = driver.findElement(By.cssSelector("#delIdCheck"));
    Actions action = new Actions(driver);
    action.moveToElement(input);
    action.click().perform();
    input.sendKeys("1");
    input.submit();
    input.findElement(By.cssSelector("button.btn:nth-child(3)"));
    action.moveToElement(input);
    action.click().perform();
```

Automation Testing – results

Package Explorer | JUnit

Finished after 11.475 seconds

Runs: 19/19 Errors: 0 Failures: 0

- runner.TestRunner [Runner: JUnit 4] (6.570 s)
  - Feature: NavigateOnChoonz (6.570 s)
    - Scenario: Navigating using the header (1.061 s)
      - Given I can access Choonz (0.007 s)
      - When user clicks on the artists tab (0.001 s)
      - Then user accecesses artists (0.002 s)
      - When user clicks on albums tab (0.001 s)
      - Then user accecesses albums (0.001 s)
      - When User clicks on playlists page (0.001 s)
      - Then user accesses playlists (0.001 s)
      - When user clicsk on tracks page (0.001 s)
      - Then user accesses tracks (0.000 s)
      - When user clicks on genres page (0.001 s)
      - Then user accesses genres page (1.045 s)
    - Scenario: entering data (1.279 s)
      - Given I can access Choonz (0.005 s)
      - Given user goes to artists page (0.001 s)
      - When user clicks on create Artist (0.001 s)
      - Then user clicks read artists (0.002 s)
      - When user updates artist (0.001 s)
      - Then user clicks read artists again (0.000 s)

The Back End

As the Back end had allready been started, it was decided that the best approach in determining what worked and what didn’t was to write the tests first before we begun writing the code. This provided the time saving benefit of simply running the tests to ensure a feature worked as intended as well as adhearing to a Test Driven Development workflow.

The Back End – Unit tests

Unit Testing is the process of testing individual units/components of code within the program to be tested. The purpose Unit testing fulfills is to ensure that each code component tested performs as it should do.

The Back End – Unit tests – testing the controllers

program when it is being run as an API. Here it is responsible for handling all the different post, fetch and put methods for our API that will be transmitted back and forth from the front end. Therefore the Unit tests for the controllers are designed to verify that the Data object we are able

The Back End – Unit tests – testing the controllers

```java
50    @Test
51    public void createAlbumTest() {
52        when(albumService.create(Mockito.any(Album.class))).thenReturn(validAlbumDTO);
53
54        ResponseEntity<AlbumDTO> response = new ResponseEntity<>(validAlbumDTO, HttpStatus.CREATED);
55
56        assertThat(response).isEqualTo(albumController.create(validAlbum));
57
58        verify(albumService, times(1)).create(Mockito.any(Album.class));
59    }
60
61    @Test
62    public void readAlbumTest() {
63        when(albumService.read()).thenReturn(albumDTOs);
64
65        ResponseEntity<List<AlbumDTO>> response = new ResponseEntity<>(albumDTOs, HttpStatus.OK);
66
67        assertThat(response).isEqualTo(albumController.read());
68        verify(albumService, times(1)).read();
69    }
70
```

The Back End – Unit tests – testing the Services

The service classes are responsible for handling the SQL data obtained through the repository into data we can handle through JAVA

Back end showcase

GET http://localhost:8090/artists/read Send

```
1 []
```

POST http://localhost:8090/artists/create

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "name":"Mario",
3     "albums": [{"name":"Mario's mxitape"}, {"name":"Mario's mixtape 2"}]
4 }
```

GET http://localhost:8090/artists/read

SELECT * FROM ARTIST;

ARTIST_ID	NAME
2	Mario

(1 row, 2 ms)

```
{
    "id": 2,
    "name": "Mario",
    "albums": [
        {
            "id": 4,
            "name": "Mario's mxitape",
            "tracks": null,
            "cover": null
        },
        {
            "id": 5,
            "name": "Mario's mixtape 2",
            "tracks": null,
            "cover": null
        }
    ]
}
```

Albums automatically created

SELECT * FROM ALBUM;

ALBUM_ID	COVER	NAME	GENRE_ID
3	*null*	Mario	*null*
4	*null*	Mario's mxitape	*null*
5	*null*	Mario's mixtape 2	*null*

(3 rows, 3 ms)

Join table automatically populated

SELECT * FROM ARTIST_ALBUM;

ARTIST_ID	ALBUM_ID
2	4
2	5

(2 rows, 2 ms)

```
@Id
@Column(name = "album_id", nullable = false)
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;

@NotNull
@Size(max = 100)
@Column(unique = true)
private String name;

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name = "album_id")
private List<Track> tracks;

@ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
@JoinTable(name = "Artist_Album",
    joinColumns = @JoinColumn(name = "album_id", referencedColumnName = "album_id"),
    inverseJoinColumns = @JoinColumn(name = "artist_id", referencedColumnName = "artist_id"))
private List<Artist> artists;

@ManyToOne
private Genre genre;

private String cover;
```

Back End Tests

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
@AutoConfigureMockMvc
@Sql(scripts = {"classpath:test-data.sql"},
        executionPhase = ExecutionPhase.BEFORE_TEST_METHOD)
@Sql(scripts = "classpath:test-teardown.sql", executionPhase = ExecutionPhase.AFTER_TEST_METHOD)
public class AlbumControllerIntegrationTest {
```

Runs: 71/71 Errors: 0 Failures: 0

- AlbumControllerIntegrationTest [Runner: JUnit 5] (3.286 s)
- TrackControllerUnitTest [Runner: JUnit 5] (2.077 s)
- PlaylistControllerUnitTest [Runner: JUnit 5] (1.884 s)
- ArtistControllerUnitTest [Runner: JUnit 5] (1.762 s)
- GenreControllerUnitTest [Runner: JUnit 5] (1.771 s)
- AlbumControllerUnitTest [Runner: JUnit 5] (1.751 s)
- PlaylistServiceIntegrationTest [Runner: JUnit 5] (1.841 s)
- GenreServiceUnitTest [Runner: JUnit 5] (0.071 s)
- GenreServiceIntegrationTest [Runner: JUnit 5] (1.797 s)
- PlaylistServiceUnitTest [Runner: JUnit 5] (0.032 s)
- AlbumServiceIntegrationTest [Runner: JUnit 5] (0.041 s)
- ArtistServiceIntegrationTest [Runner: JUnit 5] (1.933 s)
- TrackServiceUnitTest [Runner: JUnit 5] (1.727 s)
- AlbumServiceUnitTest [Runner: JUnit 5] (1.634 s)
- ArtistServiceUnitTest [Runner: JUnit 5] (0.027 s)

Element	Coverage	Covered Instru
Choonz-Starter-master	76.4 %	

Non-functional testing - JMeter

- Load Tests – Purpose of this test is to make sure the expected load can be handled
- Spike Tests – Purpose of this test is to see how well the site can handle users joining in a large amount at a specific time
- Soak Tests – Purpose is to run over a long time (normally 3 days/weekend) to see how well it handles prolonged usage
- Stress Tests – Purpose is to overload the system and see how well it copes

ANY QUESTIONS?