

October – December 2018

340CT - SOFTWARE QUALITY AND PROCESS MANAGEMENT

MY MUSIC APP

1. FUNCTIONALITY

Registration/Login

https://livecoventryac-my.sharepoint.com/:v:/g/personal/pozdnako_uni_coventry_ac_uk/EfvpTLzjz4pIkrkY5oIe_CsBkE9K1eLiu3yA2UVgSffoxA?e=08uSjY

Playlist Management

https://livecoventryac-my.sharepoint.com/:v:/g/personal/pozdnako_uni_coventry_ac_uk/EdNcPP5fyCBLlyNM92_pa7MBmGd62uBbgdja8aMoOCiLdQ?e=L2R171

Music Streaming

https://livecoventryac-my.sharepoint.com/:v:/g/personal/pozdnako_uni_coventry_ac_uk/EaONmppe-4dMu0QQjdEybukBAh_08-pGt1MsqpNzLsdgng?e=03mjd7

2. DEVELOPMENT PROCESS

System Analysis

Domain Modelling

Having learned the project requirements, the first step taken was to capture the domain knowledge and express it in a model.

A. Knowledge Crunching

In order to make the subsequent work effective, a 'knowledge crunching' session was held with the domain expert (i.e. the lab tutor – Dr Mitchell). This helped establish abstract model of the problem domain.

B. Ubiquitous Language

Also, during the session Dr Mitchell and I worked on domain language extraction. We built the domain vocabulary and split the complexity into smaller contexts.

From this work a learning was gained about the mechanics and dynamics of the domain that the software implementation had to replicate.

The output of it was the domain language description, available in Appendix A.

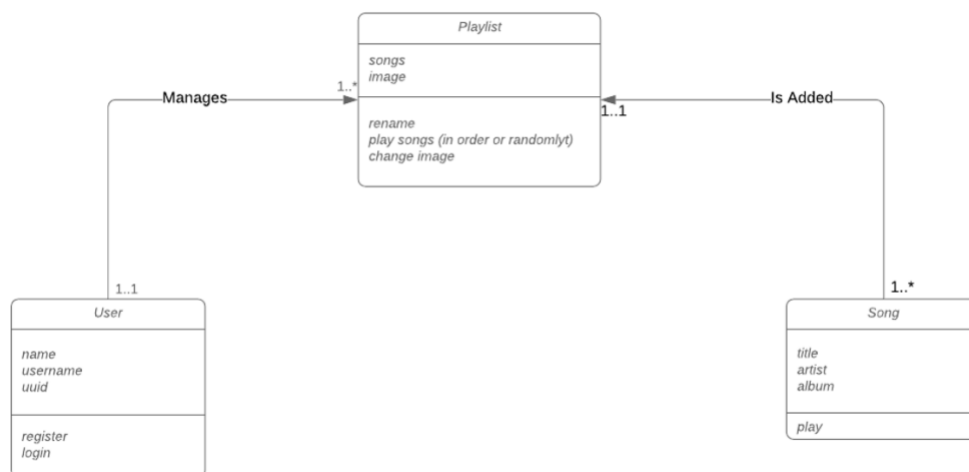
Building domain model and binding it with implementation

Having ascertained the concepts that the domain experts would use to think about problems, next, I needed to organize them explicitly into a model that software could be based on.

C. Building a domain model

The analysis then was processed to capture concepts from the domain, which could be constructed as components with software development tools, that would correctly solve the application needs.

The software system, reflecting the domain model in a literal way, was designed with very obvious mapping.

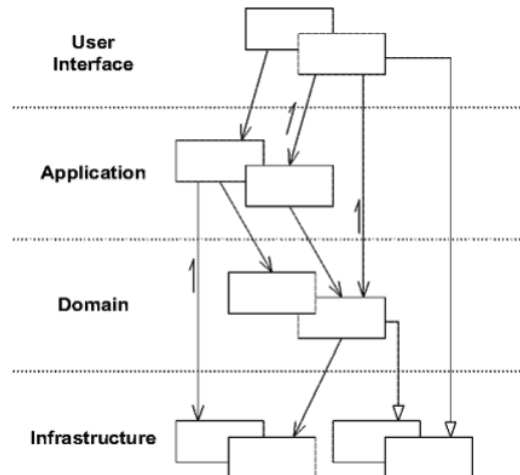


Designing the system (Architecture)

Selection of architectural patterns

D. Isolating the domain

Mr Evans (2003) suggests: “we need to be able to look at the elements of our model and see them as a system”, that would be decoupled from other functions. The standard approach of a layered architecture was employed to make the domain explicit in the mass of the system.



Model View Controller (MVC) pattern was selected to cultivate the isolation of the domain layer, enabling the conceptual classes (models) to be designed without being concerned about the user interface interacting with them.

Additionally, a modification of adding services to MVC was made to suit the need of keeping the application layer focused on its purpose and facilitated loose coupling.

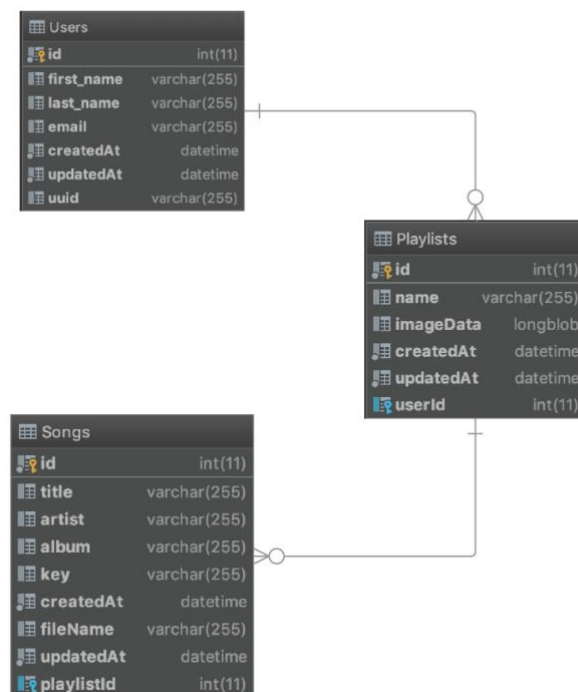
Model Expressed in Software

While producing the code, the guideline was followed that it must be an expression of the model.

i. Information System

The first step taken in creating the system design, which would reflect the domain model, was to map the conceptual classes on domain classes used in the software.

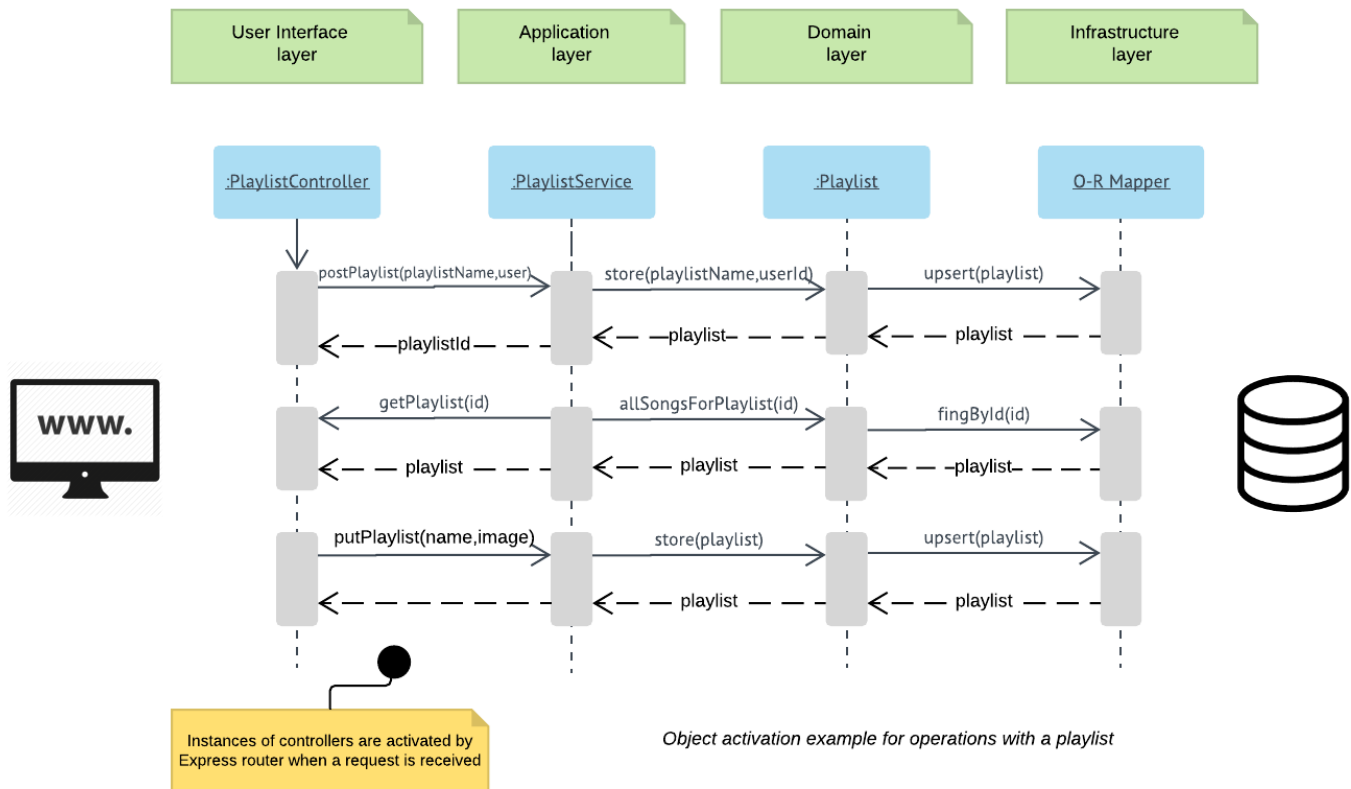
This was done by creating an ER Diagram.



ii. Object interaction

Next, to visualise how objects would interact arranged in time sequence according to the chosen MVC architectural pattern, a sequence diagram was produced.

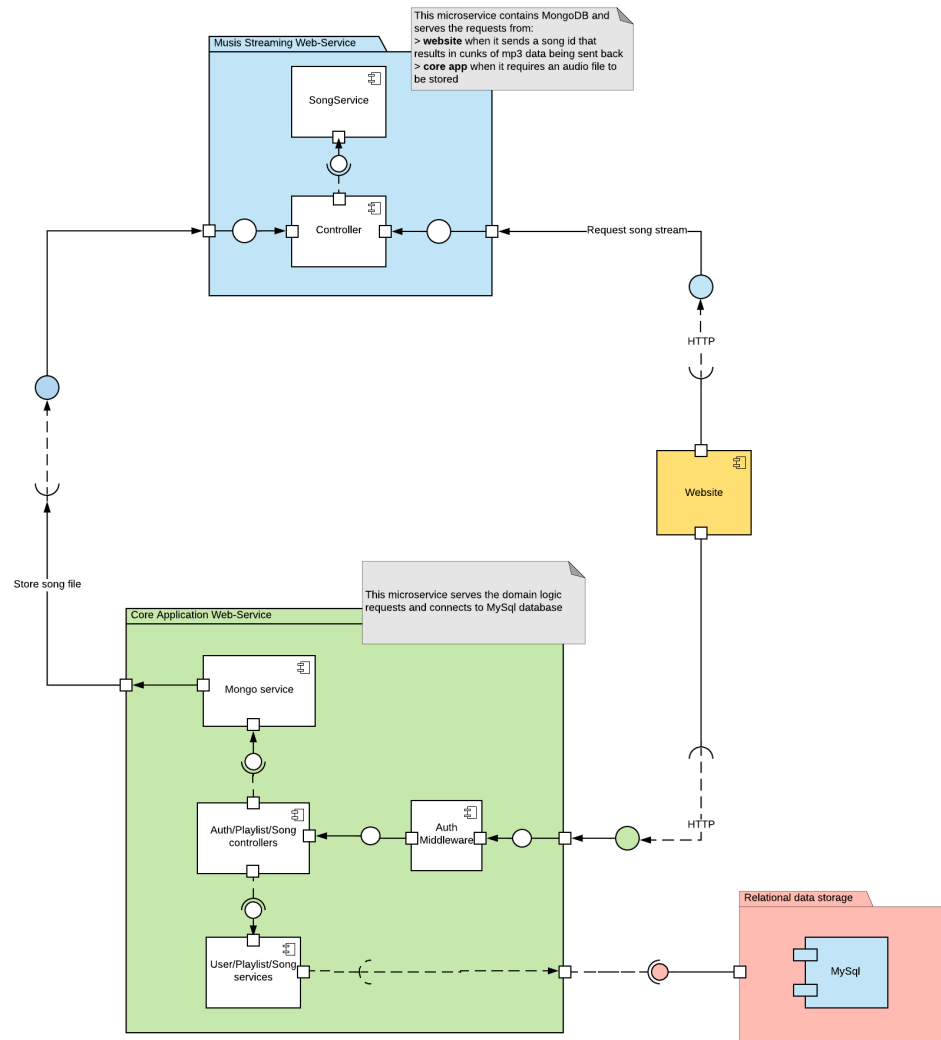
An example of object interaction and message passing in playlist related operations is provided below.



iii. High level architecture

Having gained fundamental appreciation of lower level processes, work was started on the high-level design of the system.

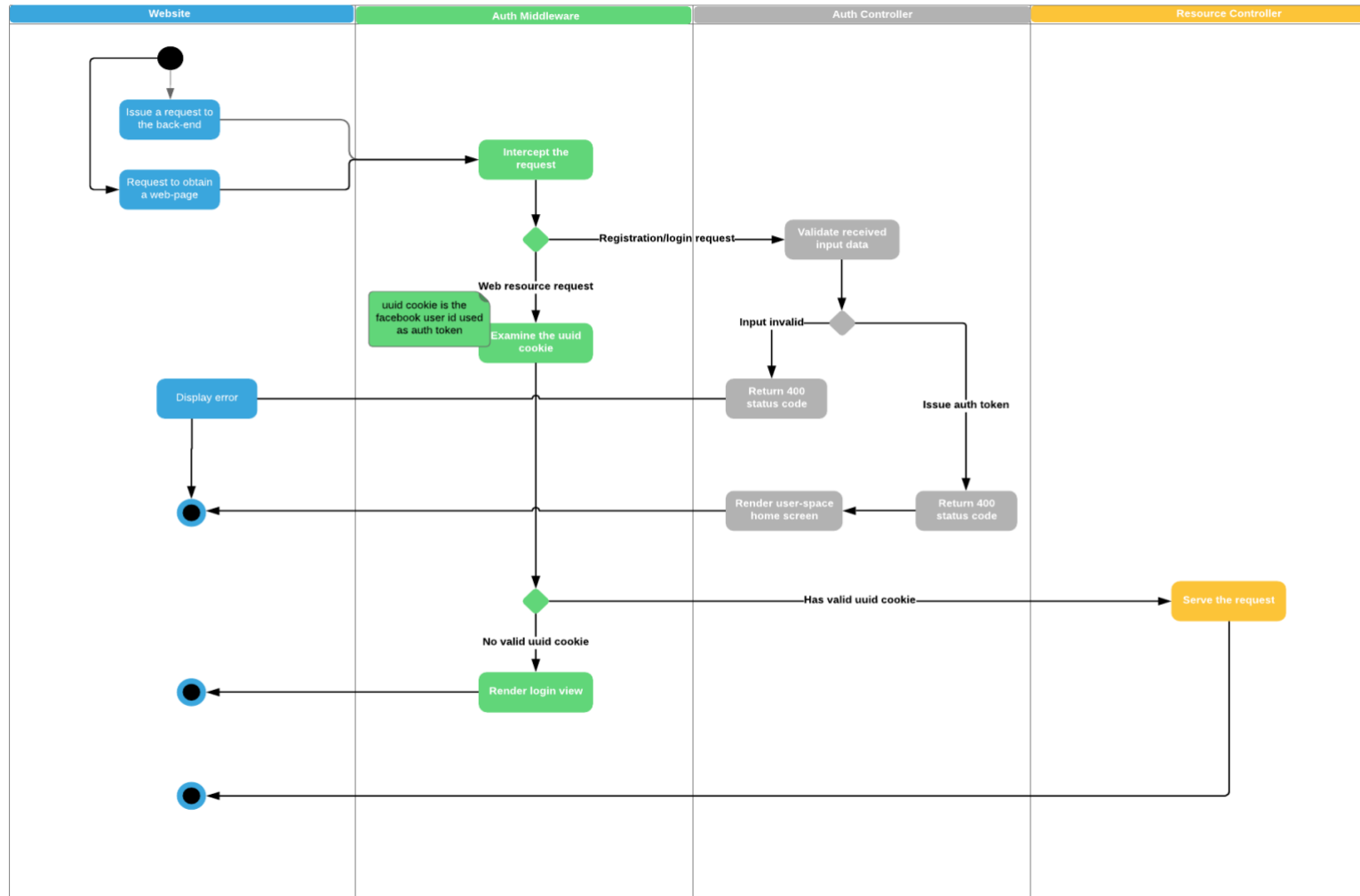
It was depicted in a component diagram which was reiterated several times. This helped the thought process of wiring several components together and ensure that all the requirements were covered.



Based on the fact that music storage and streaming are resource intensive processes, the system scalability became of a major concern. This was addressed by having dedicated databases and isolate them together with the infrastructure and functionality they served for. This also meant that the system would benefit from increased resilience and easier maintenance.

iv. Authentication & Authorization

During the development, the question arose about the authentication and authorization matters. It was resolved using the [application level middleware](#). Its implementation is depicted in the activity diagram below.



SYSTEM IMPLEMENTATION

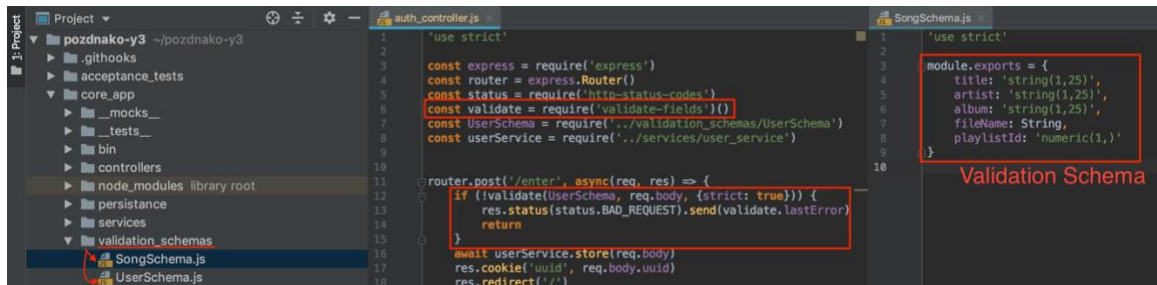
Templating

Handlebars temple engine was used to render dynamic elements on the views.

The user interface was built employing Bootstrap, jQuery and other libraries to enhance user experience.

Input validation

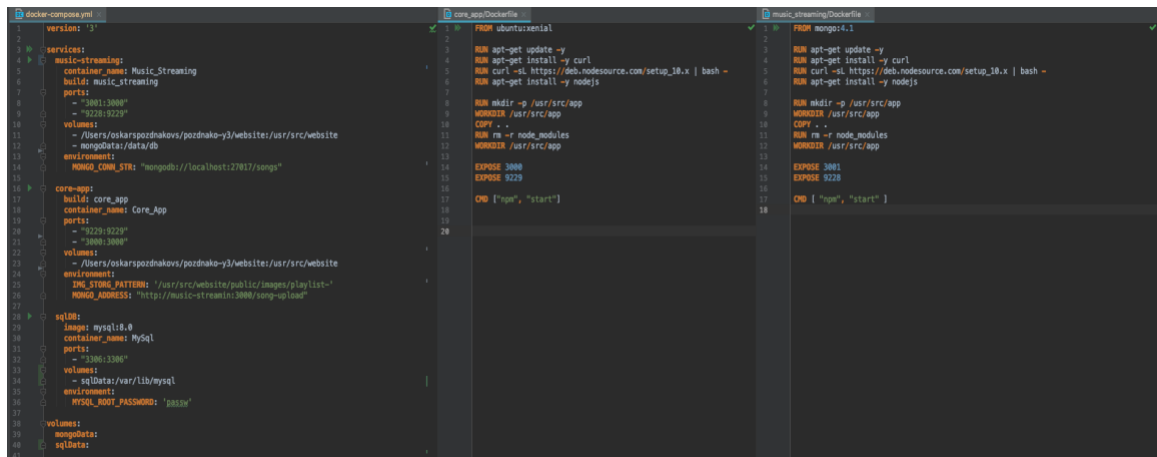
An npm library was added for input validation.



Microservices

Docker was used to implement the Microservice architecture and the following services were defined:

1. **Core App** serves the domain logic requests.
2. **Music Streaming** uses MondoDB to store and stream audio files and serve corresponding requests.
3. **SQL Database** to store the relational data. This was encapsulated in a different container because it was intended to also add [Auth middleware](#) to **Music Streaming** that would require access to it. Due to lack of time it was not done.

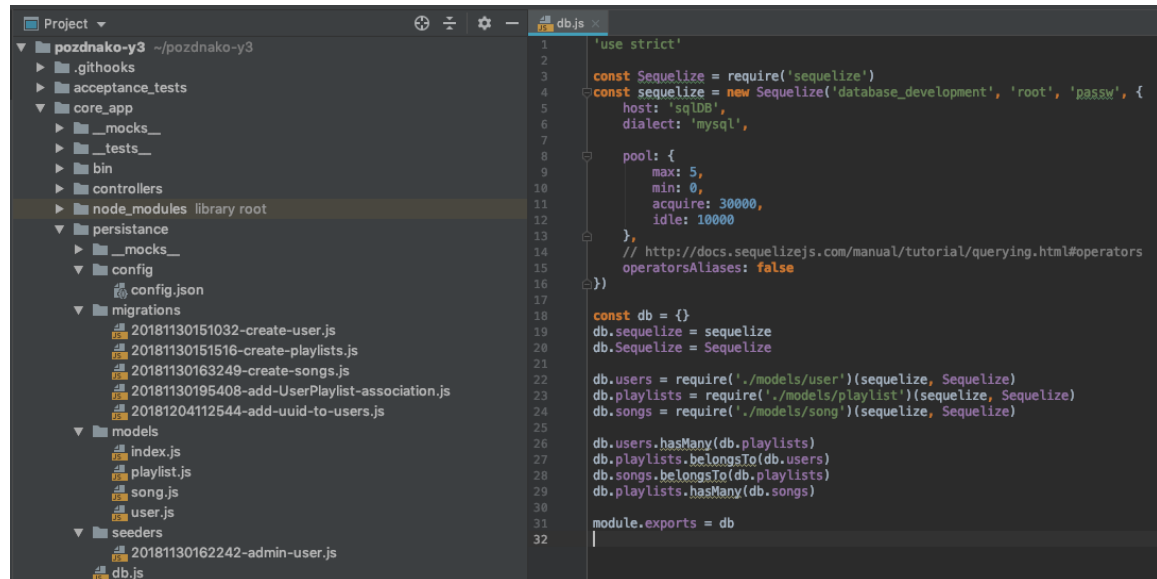


Object Relational Mapping (ORM)

The system has an integrated ORM library: [sequelize](#).

Its features that were utilised are as follows: migrations, seeds and object mapping.

Also, a centralised model access object (db.js) was created and exported.



The screenshot shows a code editor with a project structure on the left and the content of `db.js` on the right. The project structure includes folders like `core_app`, `node_modules`, `persistence`, `migrations`, `models`, and `seeders`. The `db.js` file contains the following code:

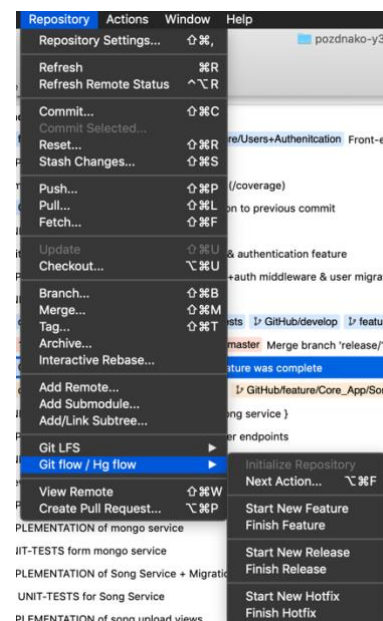
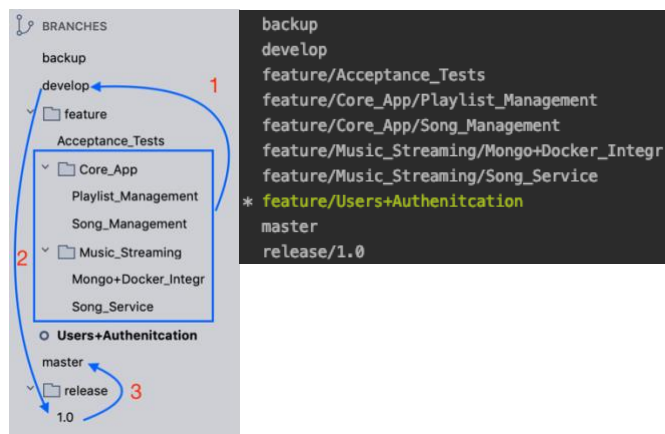
```
1 'use strict'
2
3 const Sequelize = require('sequelize')
4 const sequelize = new Sequelize('database_development', 'root', 'password', {
5   host: 'sqlDB',
6   dialect: 'mysql',
7
8   pool: {
9     max: 5,
10    min: 0,
11    acquire: 30000,
12    idle: 10000
13  },
14  // http://docs.sequelizejs.com/manual/tutorial/querying.html#operators
15  operatorsAliases: false
16 })
17
18 const db = {}
19 db.sequelize = sequelize
20 db.Sequelize = Sequelize
21
22 db.users = require('./models/user')(sequelize, Sequelize)
23 db.playlists = require('./models/playlist')(sequelize, Sequelize)
24 db.songs = require('./models/song')(sequelize, Sequelize)
25
26 db.users.hasMany(db.playlists)
27 db.playlists.belongsTo(db.users)
28 db.songs.belongsTo(db.playlists)
29 db.playlists.hasMany(db.songs)
30
31 module.exports = db
32
```

The migrations and the seed are executed when the package script is run in the **Core App**.

Code Quality

A static code analysis tool ESLint was used to enforce rules ensuring cleaner syntax and the subset of JavaScript used that does not compromise the code quality.

Version Control



Git Flow branching model was used for the version control. It was done with the help of the tools integrated in my git GUI after Git Flow was initialised in the repository.

The system was developed by creating feature branches that upon completion were pulled into *develop* via PRs. When all the *Core_App* and *Music_Streaming* related branches were in *develop*, the *release/1.0* were branched off and subsequently a PR was completed to merge it into *master*.

Git log --graph was used to visualise dependency tree for all commits

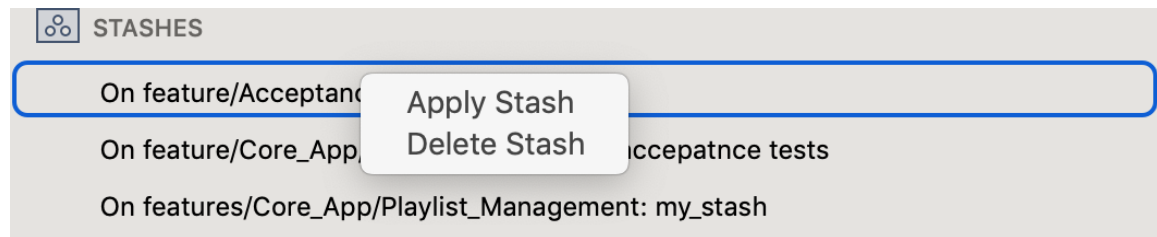
```
Oskarss-MBP:pozdnako-yyy Oscar$ git log --graph
* commit f5f113ced2275da838b0c2e9056964501bfb218b (HEAD -> master, tag: 1.0, origin/master, origin/HEAD)
  \ Merge: dfe719f e240c07
  | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  | Date: Tue Dec 4 21:28:23 2018 +0000
  |
  | Merge branch 'release/1.0'
  |
  | * commit e240c074fab27d44ec18e3a824c84e12af5237c1
  |   \ Merge: 893da56 128bb31
  |     | Author: Oskars Pozdnakovs (pozdnako) <pozdnako@coventry.ac.uk>
  |     | Date: Tue Dec 4 21:21:55 2018 +0000
  |     |
  |     | Song feature was complete
  |     |
  |     | Song feature was complete
  |     |
  |     | * commit 128bb3113ce271c0af68c4507506e01a7c00394a (origin/feature/Core_App/Song_Management)
  |     |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   | Date: Tue Dec 4 11:21:08 2018 +0000
  |     |   |
  |     |   | IMPLEMENTATION of get all songs for user [song service]
  |     |   |
  |     |   | * commit 39c9c09bb7099d7a940d0c3b504ced1e015636e3
  |     |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   | Date: Tue Dec 4 11:06:51 2018 +0000
  |     |   |   |
  |     |   |   | UNIT-TESTS for getting all songs for user {song service }
  |     |   |   |
  |     |   |   | * commit 44931d3c59650e01281862ee3267ab06525621a2
  |     |   |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   |   | Date: Tue Dec 4 10:51:49 2018 +0000
  |     |   |   |   |
  |     |   |   |   | IMPLEMENTATION of: obtain all songs for user endpoints
  |     |   |   |   |
  |     |   |   |   | * commit 46c4ebc3586e0f4f6b42fc9230983e6bb403be05
  |     |   |   |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   |   |   | Date: Tue Dec 4 10:21:27 2018 +0000
  |     |   |   |   |   |
  |     |   |   |   |   | UNIT-TESTS to obtain all songs for users
  |     |   |   |   |   |
  |     |   |   |   |   | * commit 4fa6c1be46baa596fdc1b04239bc4cf86b979377
  |     |   |   |   |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   |   |   |   | Date: Mon Dec 3 23:57:22 2018 +0000
  |     |   |   |   |   |   |
  |     |   |   |   |   |   | Views for all songs preview
  |     |   |   |   |   |   |
  |     |   |   |   |   |   | * commit 7fb854f8715391cf3952b89bd3a5e561e6fa6ab9
  |     |   |   |   |   |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   |   |   |   |   | Date: Mon Dec 3 21:28:47 2018 +0000
  |     |   |   |   |   |   |   |
  |     |   |   |   |   |   |   | IMPLEMENTATION of song playback controls (gui)
  |     |   |   |   |   |   |   |
  |     |   |   |   |   |   |   | * commit 82b179153fe5af01f7bf6af0d36bf8ada6f8ea68
  |     |   |   |   |   |   |   |   | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  |     |   |   |   |   |   |   |   | Date: Mon Dec 3 16:31:28 2018 +0000
```

Git tag, occasionally annotated tags were added to commits that were followed by pull requests.

```
Oskarss-MBP:pozdnako-yyy Oscar$ git log --graph
* commit f5f113ced2275da838b0c2e9056964501bfb218b (HEAD -> master, tag: 1.0, origin/master, origin/HEAD)
  \ Merge: dfe719f e240c07
  | Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
  | Date: Tue Dec 4 21:28:23 2018 +0000
  |
  | Merge branch 'release/1.0'
```

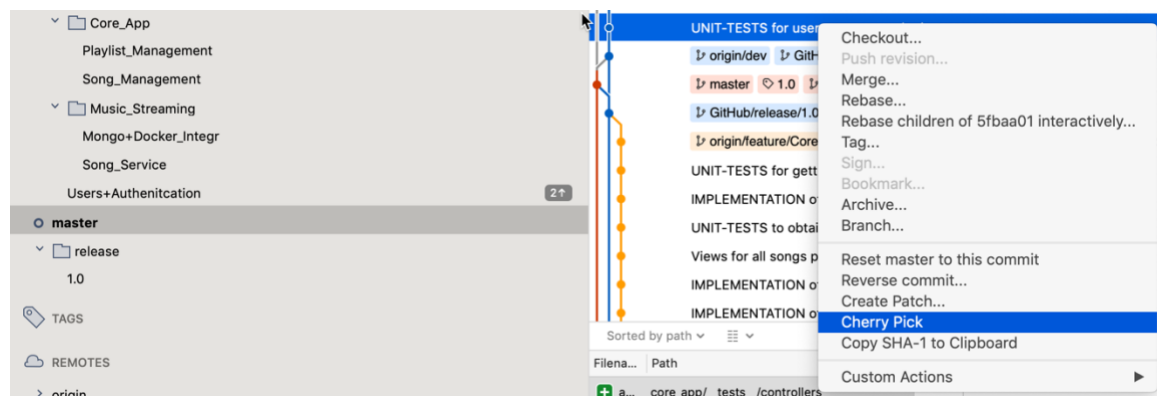
Git stash

Stashing was occasionally performed when I needed to switch between incompatible branches, but I was constrained by the recent commits.



Cherry picking

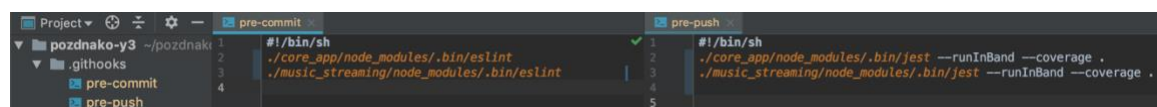
It was useful when I needed to obtain a particular commit from a different branch that didn't have it.



Continuous Integration

GitHooks were also used to ensure continuous quality of the software.

- pre-commit – run ESLinter
- pre-push – run the Jest test suites



The configuration for successful test completion can be seen here -->

```
"jest": {
  "testEnvironment": "node",
  "verbose": true,
  "coverageThreshold": {
    "global": {
      "branches": 80,
      "functions": 80,
      "lines": 80,
      "statements": -10
    }
  },
  "collectCoverageFrom": [
    "**/*.jsx",
    "!**/node_modules/**",
    "!**/coverage/**",
    "!**/bin/www/**",
    "!**app.js**",
    "!**/persistence/**"
  ]
}
```

3. TESTING

TDD

Test Driven Development approach was applied most of the time working on this project.

It was used in the following way:

1. an automated test case was written that defined a desired improvement or functionality
2. the entire test suite was run to ensure the test case failed
3. then minimum amount of code was written to pass that test case(s)
4. the entire test-suite was re-run to see if the new test case passed

Because the architecture had a service layer, when a controller unit test was written a mocked service was created and committed along.

In the example below, when the implementation of the controller was committed, it was using mocked song_service.js until the actual one was implemented subsequently.

This screenshot shows a commit diff for the implementation of obtaining all songs for user endpoints. The commit message is "UNIT-TESTS to obtain all songs for users". The diff highlights the file `core_app/_tests_/controllers/song_controller.test.js`. The code snippet shows a test case for the GET /songs endpoint, expecting a 200 status and that the `ss.allSongsForUser` method has been called.

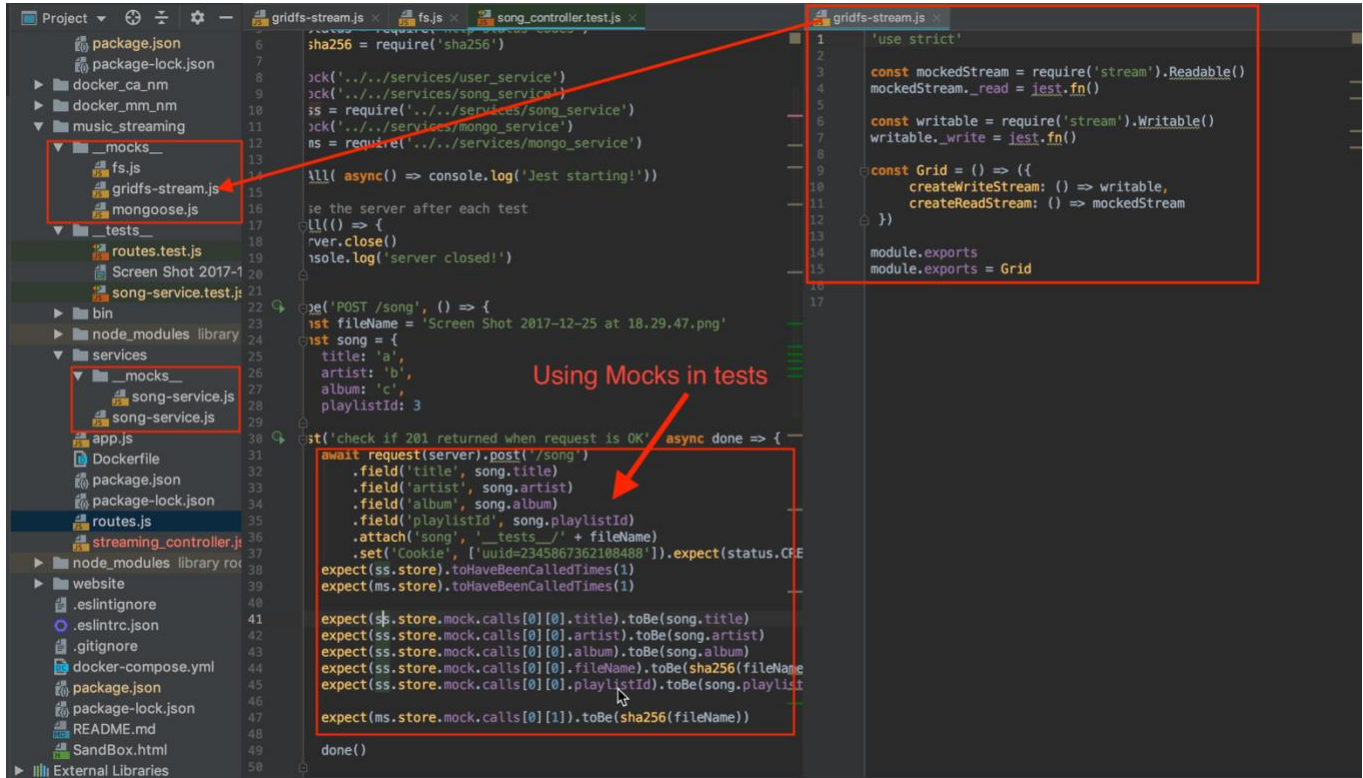
```
IMPLEMENTATION of: obtain all songs for user endpoints
UNIT-TESTS to obtain all songs for users
Sorted by path
Filename Path
song_controller.test.js core_app/_tests_/controllers
Hunk 1: Lines 80-91
80 80 done()
81 81 })
82 82 })
83 +
84 + describe('GET /songs', () => {
85 +   test('check if 200 endpoint is served', async done => {
86 +     await request(server).get('/songs')
87 +     .expect(status.OK)
88 +     expect(ss.allSongsForUser).toHaveBeenCalled()
89 +     done()
90 +   })
91 + })
Commit: 46c4ebc3586e0f4f6b42fc9230983e6bb403be05 [46c
Parents: 4fa6c1be46
Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
```

This screenshot shows a commit diff for the implementation of obtaining all songs for user endpoints. The commit message is "IMPLEMENTATION of: obtain all songs for user endpoints". The diff highlights the file `core_app/controllers/song_controller.js`. The code snippet shows the implementation of the GET /songs endpoint, which calls `songService.allSongsForUser(1)` and renders the response.

```
IMPLEMENTATION of: obtain all songs for user endpoints
UNIT-TESTS to obtain all songs for users
Sorted by path
Filename Path
song_controller.js core_app/controllers
song_service.js core_app/services/_mocks_
scrsh.png website
Hunk 1: Lines 30-43
30 30 })
31 31 })
32 32
33 + router.get('/songs', async(req, res) => {
34 +   const songs = await songService.allSongsForUser(1)
35 +   res.render('songs_body', {songs: songs})
36 + })
37 +
38 + router.get('/login', async(req, res) => {
39 +   res.render('blank', {layout: 'login.hbs'})
40 + })
41 +
42 + module.exports = router
43 +
Commit: 44931d3c59650e01281862ee3267ab06525621a2 [4
Parents: 46c4ebc358
Author: Oskars Pozdnakovs <opozdnakov@uni.coventry.ac.uk>
```

Mocks

The system has a complete set of mocks for external and internal dependencies. Some mocks are manual, and some are automatic. Altogether 9 dependencies were mocked for testing and TDD purposes.



Code Coverage

Core App

Test Results	511m
song_controller.test.js	150m
POST /song	100m
check if 201 returned when request is OK	52m
check if 400 when no album	20m
check if 400 when playlistid has wrong format	18m
check if 500 returned when service throws	13m
GET /songs	60m
check if 200 endpoint is served	60m
playlist_controller.test.js	268m
GET /playlists	78m
test if returns 200 when requested	78m
GET /playlistid	48m
check if view returned if id given	29m
check if 400 if id invalid	17m
POST /playlist	44m
check if 201 returned when request is OK	30m
check if 400 when no name	11m
check if 500 returned when service throws	3m
PUT /playlistid	73m
DELETE /playlistid	17m
auth_controller.test.js	66m
POST /enter	25m
get /logout	31m
song_service.test.js	3m
Song Service tests	3m
user_service.test.js	20m
User Service test	20m
playlist_service.test.js	1m
Playlist service test suit	16m
mongo_service.test.js	9m
Mongo Service tests	9m
check if 400 when no album	9m

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	93.2	82.14	84.62	95.07	
core_app	92.31	87.5	100	92.31	19
auth_middleware.js	92.31	87.5	100	92.31	
core_app/controllers	98.84	93.75	92.31	98.84	
auth_controller.js	100	100	100	100	
playlist_controller.js	100	91.67	100	100	48
song_controller.js	96.43	100	80	96.43	39
core_app/services	82.61	25	75	87.8	
mongo_service.js	66.67	0	50	80	17
playlist_service.js	92.31	100	75	100	
song_service.js	78.95	100	75	83.33	10,19,28
user_service.js	87.5	50	100	85.71	12
core_app/validation_schemas	100	100	100	100	
SongSchema.js	100	100	100	100	
UserSchema.js	100	100	100	100	

Test Suites: 7 passed, 7 total
Tests: 30 passed, 30 total

Music Streaming

Test Results	96 ms	jest --coverage --runInBand
routes.test.js	82 ms	console.log __tests__/routes.test.js:9
POST /song-upload	62 ms	Jest starting!
check if 202 returned when sending file	62 ms	POST /song-upload 202 5.854 ms --
GET /music_streaming/id	20 ms	GET /song1 200 4.675 ms --
check if 200 returned when requesting a stream	20 ms	console.log __tests__/routes.test.js:14
song-service.test.js	13 ms	server closed!
SongService test suit	13 ms	(node:17626) UnhandledPromiseRejectionWarning: Unhandled promise rejection (rejection id: 2): undefined
test if finish event is called	2 ms	(node:17626) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future,
test if error event is called	5 ms	
check if .openReadStream() gets to streaming finish	6 ms	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	95.45	100	92.86	95.45	
music_streaming	95.24	100	100	95.24	
app.js	100	100	100	100	
routes.js	90	100	100	90	17,29
music_streaming/services	95.83	100	98	95.83	
song-service.js	95.83	100	98	95.83	35

Acceptance tests

Testcafe framework was used to perform front-end testing.

Two features have such test suites:

- Sign-in/Sign-up (2 test cases)
- Playlist management (4 test cases)

The image shows a VS Code editor with two files open: `playlists.js` and `login_logout.js`. The left sidebar displays a file explorer for a project named `pozndnako-y3`, showing a directory structure with `acceptance_tests`, `core_app`, `docker`, `music_streaming`, `node_modules`, and `website`. The main editor area shows the code for both files, which are Jest test suites for a web application.

playlists.js

```

1 'use strict'
2 import { Selector } from 'testcafe';
3
4 fixture('Playlist operation tests')
5   .page('http://localhost:3000');
6
7 const playlistName = 'TEST'
8
9 test('Playlist can be created', async (t) => {
10   // search github
11   await t
12     .click('#LinkedInLogin')
13     .click('ul.nav.nav-pills.nav-stacked.custom-nav li:nth-cl
14     .click('#page-wrapper > div.inner-content > div.music-br
15     .click('#page-wrapper > div.inner-content.single > div.s:
16
17   const playlistBarSongs = Selector('#playlist > div')
18
19   await t
20     .expect(playlistBarSongs.count).eq(1)
21
22   test('Test playlist deletion', async (t) => {
23     // search github
24     await t
25       .click('#LinkedInLogin')
26       .click('ul.nav.nav-pills.nav-stacked.custom-nav li:nth-cl
27       .click('#page-wrapper > div.inner-content > div.music-br
28       .click('#deletePlaylist')
29
30     const newPlaylistBtn = Selector('#newPlaylist')
31
32     await t
33       .expect(newPlaylistBtn.exists).ok()

```

login_logout.js

```

1 'use strict'
2 import { Selector } from 'testcafe';
3
4 fixture('Login/Logout Test')
5   .page('http://localhost:3000');
6
7 test('LinkedIn login takes us to the user-space', async (t) => {
8   // search github
9   await t
10     .click('#LinkedInLogin')
11
12   const exitBtnExists = await Selector('ul.nav.nav-pills.nav-st
13
14   await t
15     .expect(exitBtnExists).eq(4);
16
17   test('Upon logout user is redirected to login page', async (t) => {
18     // search github
19     await t
20       .click('#LinkedInLogin')
21       .click('#exit')
22
23     const exitBtnExists = await Selector('#LinkedInLogin').exists
24
25     await t
26       .expect(exitBtnExists).ok();

```

```
Oskarss-MBP:pozdnako-y3 Oscar$ testcafe safari ./acceptance_tests
```

Running tests in:

- Safari 12.0.1 / Mac OS X 10.14.1

Login/Logout Test

- ✓ LinkedIn login takes us to the user-space

- ✓ Upon logout user is redirected to login page

Playlist operation tests

- ✓ Playlist can be created

- ✓ Playlist can be renamed

- ✓ Added songs appear in playlist song list

- ✓ Test playlist deletion

6 passed (46s)

4. SUMMARY

While doing the system analysis phase with the DDD approach, I gained a learning about its usefulness when it's put in practice.

Ubiquitous language of the domain could potentially leverage clearer communication between stakeholders by eliminating all the jargon used by each party.

As DDD complements the object-oriented paradigm (OOP) with its expression of the domain model, my belief is that it would also make a system more adaptable to change. This would be because the model is almost directly mapped on objects.

Also, I have learned how to use a range of tools to ensure code quality. Among such tools are ESLint, Git hooks and CI.

I managed to familiarise with more advanced branching model than that I had used before, namely GitHub-flow.

Github-flow with its simple branching model is perfectly suitable for single developer projects with one version in production.

On the other hand, Git-flow is better suitable for projects with several versions in production and CI/CD pipelines configured.

Another powerful practice was TDD. I realised that it acts as a safety net when changes are made to the source code. Not to mention the effect it makes to force you to think of a better design in advance. However, a downside of it is that you are left with a massive number of tests to maintain, which takes a lot of time.

I could possibly have done more work on the acceptance testing making snapshots in headless browsers and add to CI. This would massively improve the software quality.

APPENDIX A

Users can **sign in**, or **register** if they don't have an account.

Once signed in, they can **manage their list of playlists**.

Every **playlist will have an image** and an **option to change it**.

Users can **manage song files** for a selected playlist.

Songs can be played either **in order** or **randomly**.

Registration

The users will be required to provide:

- Full name
- Phone number
- Email
- Facebook

Sign-in

Users can sign in using

- Facebook

Playlist Management

Users will be able to carry out the following actions with playlists:

- Create
- Delete
- Rename

Playlist image

The following operations will be allowed

- Upload from PNG file

Manage songs

- Upload from mp3 file

Song playback

- Change volume

- Fast forward
- Rewind
- Play/ Pause