

Harmony Release Summary

Team Members

Name and email	GitHub Id	Role of each member and tasks done by each member (brief description)
Alexander Senden sendena@myumanitoba.ca	61121238 (alexsenden)	(Architecture Designer, Regression Test Lead) Designed and implemented overall project architecture. Made over 200 tests for testing in CI.
Edden Wong wonge5@myumanitoba.ca	32918555 (EddenWong)	(Frontend Design Lead) Created website design. Made website UI consistent, and seamless.
Andrii Provozin provozia@myumanitoba.ca	54965144 (Developik)	(Deployment Manager, Load Test Lead) Managed production cloud services. Created load tests that made sure Harmony was efficient.
Christopher Baran baranc@myumanitoba.ca	91697216 (Dankgen)	(Database Administrator) Created database schema design. Made sure of efficient use of connections between the API and database.
Ikram Khan shiponik@myumanitoba.ca	33215354 (moiikram)	(Frontend Implementation Manager) Implemented features, and user stories into Harmony's frontend. Ensured that changes to

		the frontend would not deviate from desired behaviour.
--	--	--

Project Summary

1) High Level Summary

Harmony is a social media-inspired platform where users can post polls, reviews, and discussions. Every post must relate to either a song, an album, or an artist. Harmony's users can follow other users, and additionally can follow songs, albums and artists to stay up-to-date on discussion. Harmony features two "endless" feeds - one for trending posts, and one for posts by followed users/topics - so that readers have a seamless post viewing experience.

2) Differences With Proposal

Harmony's only deviation from its initial vision is that the platform has a mobile version. When the plans were first laid out, it was agreed that only a desktop version would be focused on. Upon further inspection, the website needed minimal changes to support a mobile version.

GitHub Repository Link

<https://github.com/alexsenden/harmony>

DockerHub Repository link

Note: The project is hosted publicly via Azure at <https://harmonysocial.ca>
Feel free to test on this version since it contains more posts and users than a new local instance does.

1) DockerHub Link: <https://hub.docker.com/u/aprovozin>

- a) aprovozin/harmony-fe-cd and aprovozin/harmony-api-cd are the images that are run on our cloud servers
- b) aprovozin/harmony-fe-cd-local and aprovozin/harmony-api-cd-local are the images that can be run locally
- c) There are slight configuration differences between the two, which is why they must be separate

2)

- a) Clone the GitHub repo (<https://github.com/alexsenden/harmony>), or alternatively, copy the `docker-compose.yml` file at the root of the project (<https://github.com/alexsenden/harmony/blob/main/docker-compose.yml>)
- b) Navigate to the project root/location of the `docker-compose.yml`

- file, and run the command ``docker-compose up -d``
- c) The project is now running. **The frontend is accessible at localhost:8080**, the backend is accessible at localhost:8081, and the database is accessible at localhost:5432.
 - d) Please note that the backend takes about a minute to start the first time since it needs to seed the database with the music metadata dataset.

List of User Stories

Sprint 1

[Non-technical sprint]

Sprint 2 [[Milestone Link](#)]

US #1: View Review Posts	[Done]	Link
US #2: View Poll Posts	[Done]	Link
US #3: View Discussion Posts	[Done]	Link
US #4: View Isolated Post	[Done]	Link
US #5: Following Feed	[Done]	Link
US #6: View Another User's Profile Information	[Done]	Link
US #7: Trending Feed	[Done]	Link
US #8: Follow Other Users	[Done]	Link
US #9: View All of a User's Post from Their Profile	[Done]	Link
US #10: Sign in to an Account	[Done]	Link
US #11: Create an Account	[Done]	Link
US #12: Discussions	[Done]	Link
US #13: Reviews	[Done]	Link
US #14: Polls	[Done]	Link
T #1: Create script to insert dataset into database	[Done]	Link
T #2: Add `Text` Component	[Done]	Link

Sprint 3 [[Milestone Link](#)]

US #15: Vote on Polls	[Done]	Link
US #16: Follow an Artist/Album/Song	[Done]	Link
US #17: See All Posts About Song on Topic Profile	[Done]	Link
US #18: See All Posts About Album on Topic Profile	[Done]	Link
US #19: See All Posts About Artist on Topic Profile	[Done]	Link
US #20: Change My Account Information	[Done]	Link
US #21: Search Users	[Done]	Link
US #22: Search Artists	[Done]	Link
US #23: Searching Songs	[Done]	Link
US #24: Searching Albums	[Done]	Link
US #25: Liking Posts	[Done]	Link
US #26: Commenting on Posts	[Done]	Link
T #3: Separate unit and integration tests	[Done]	Link

Sprint 4 [[Milestone Link](#)]

T #4: Change poll buttons	[Done]	Link
T #5: Set up basic load testing for Front end	[Done]	Link
T #6: Finish and import dataset	[Done]	Link
T #7: Create an "auto-login" url for presentation	[Done]	Link

T #8: “Enter” key should submit for login and signup [Done] [Link](#)
 T #9: Auto-refresh the page after creating a post [Done] [Link](#)
 T #10: Check titles for all pages [Done] [Link](#)
 T #11: Handle outliers in posts and comments [Done] [Link](#)
 T #12: Render newlines in posts [Done] [Link](#)
 T #13: Don't fetch all likes on page load [Done] [Link](#)
 T #14: Stop using custom headers for regular data [Done] [Link](#)

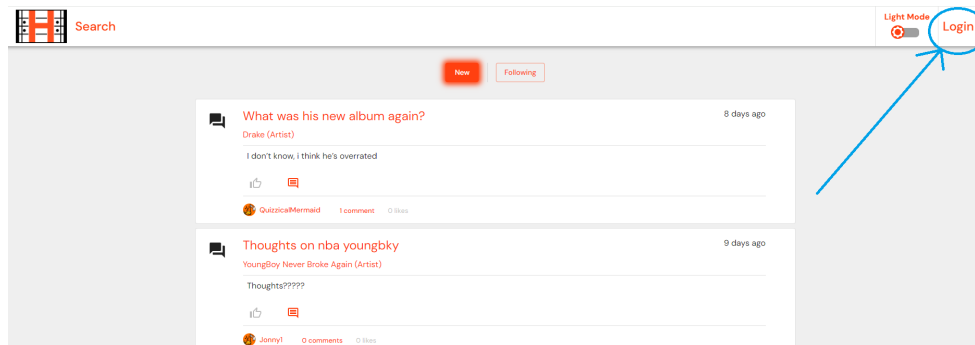
Release

User Manual

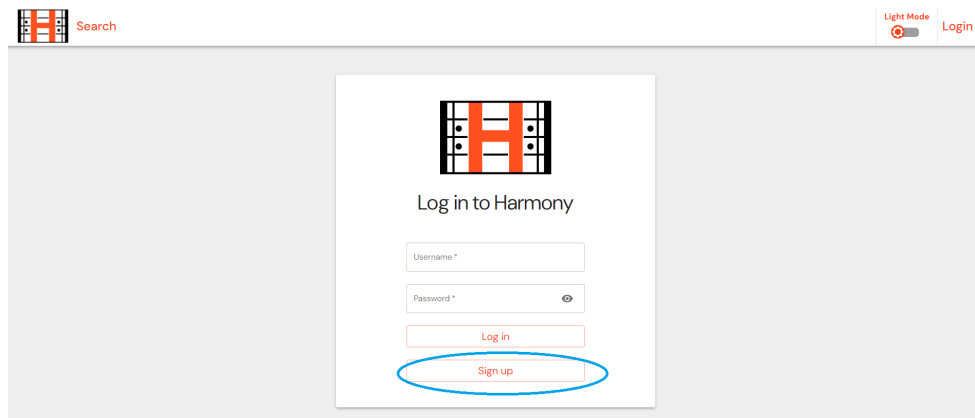
Account Management

Registration:

Step 1. While not logged in, click on the “Login” button on the right side of the app bar.



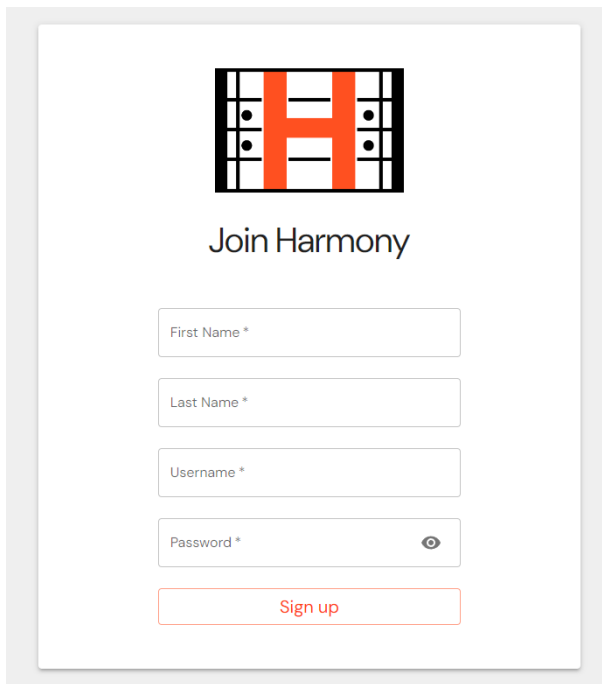
Step 2. Click on the “Sign Up” button on the bottom of the login box



Step 3. Fill out the registration information with the following requirements and click the “Sign Up” button:

- Password must contain be at least 8 characters long, have at least one upper and lower case letter (a-z, A-Z), a number (0-9), and a symbol (@, #, \$, %, ^, &, -, +, =, (,), !, ?, ' ', "),
- First and Last Name can only contain letters, spaces, -, or '

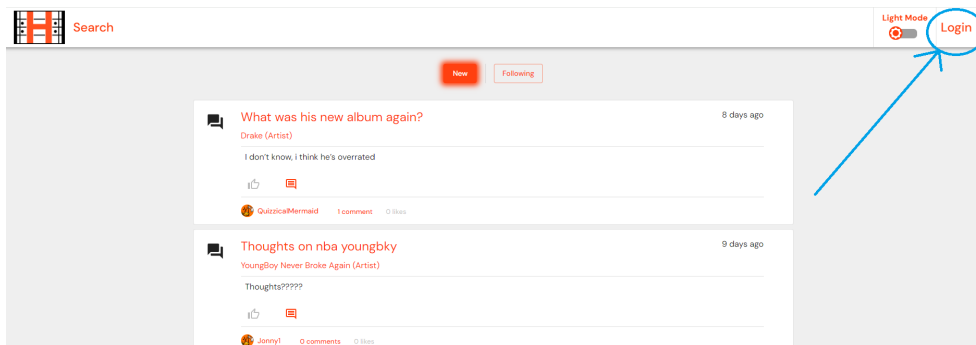
- Username must be unique



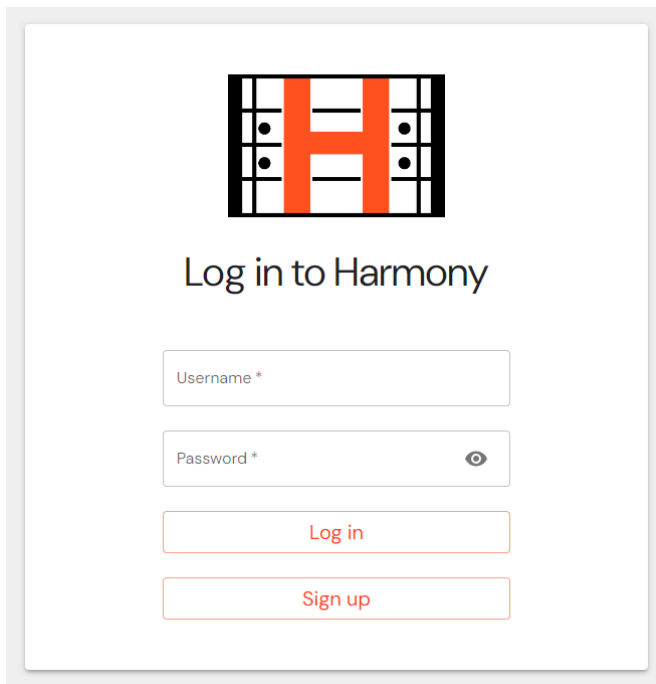
A sign-up form titled "Join Harmony" with a logo featuring a stylized orange 'H' on a black grid. The form contains four input fields: "First Name *", "Last Name *", "Username *", and "Password *". The "Password *" field has an eye icon for toggling visibility. Below the fields is a red "Sign up" button.

Login:

Step 1. While not logged in, click on the “Login” button on the right side of the app bar.



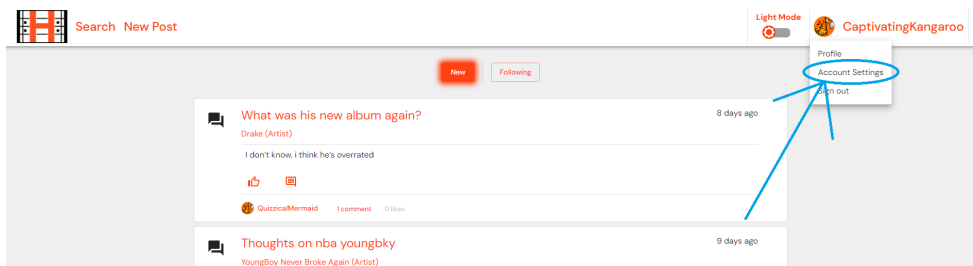
Step 2. Fill out valid account information and click the “Log in” button



The image shows the Harmony login interface. At the top is the Harmony logo, a stylized 'H' in orange and black. Below the logo is the text "Log in to Harmony". There are two input fields: "Username *" and "Password *". The password field has an eye icon to toggle visibility. Below the input fields are two buttons: "Log in" and "Sign up".

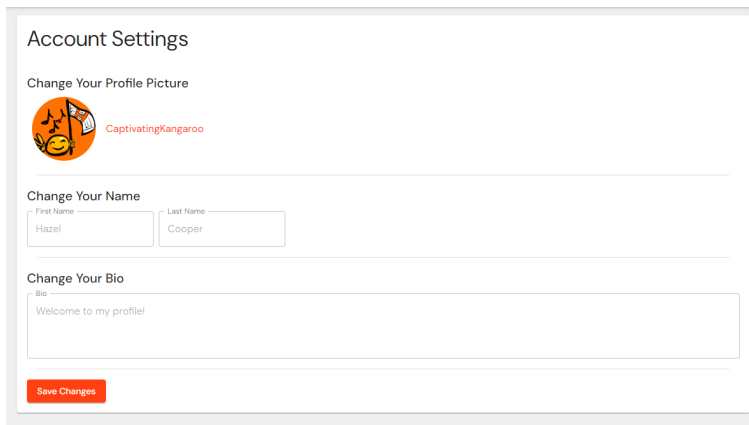
Change Account Details:

Step 1. While logged in, click on the right side of the app bar, where your user name is, and click on the “Account Settings” button that appears.




Step 2. Here, the user can click on the profile picture and select a new one, or change their first/last name with the same restrictions as when creating the account, and change

their bio that appears on their profile.



Account Settings

Change Your Profile Picture

 CaptivatingKangaroo

Change Your Name

First Name: Hazel Last Name: Cooper

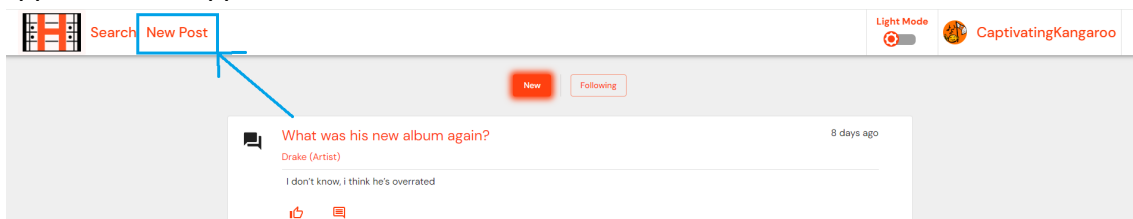
Change Your Bio

Bio: Welcome to my profile!

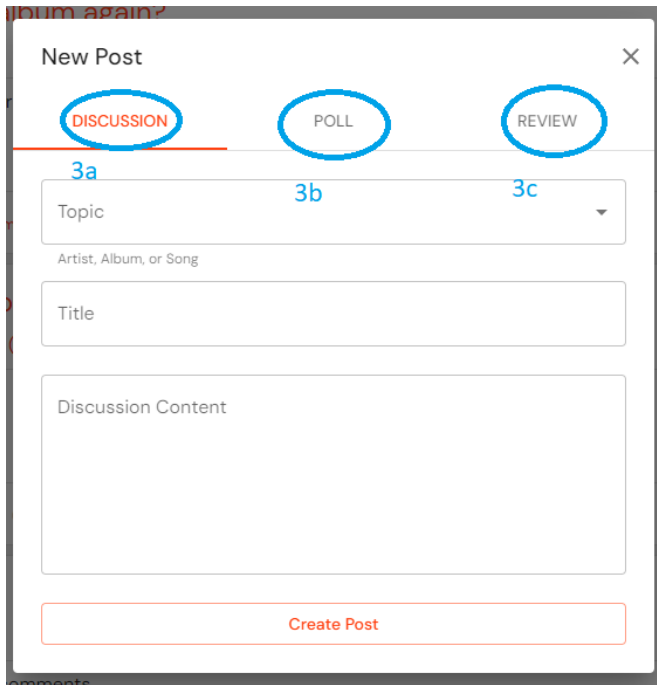
Save Changes

Creating Posts

Step 1. User must be logged in to create posts, while logged in a “New Post” button will appear on the app bar, click this.



Step 2. This opens a modal that lets users create posts, from here, go to steps 3a, 3b and 3c to see how to create discussion posts, review posts, and polls.



New Post

DISCUSSION POLL REVIEW

3a 3b 3c

Topic

Artist, Album, or Song

Title

Discussion Content

Create Post

Discussion Posts:

Step 3a. Fill out the topic by typing in the name of an artist, album, or song. If the entry exists on the database then the user will be given topics that match what they typed and they can select a topic.

New Post

DISCUSSION POLL REVIEW

Topic

taylor sw

- Taylor Swift (Artist)
- Taylor Swift (Album)
- Taylor Swift Music History (Song)

Discussion Content

Create Post

Step 4a. Fill out a title for your post, this can be up to 150 characters long

New Post

DISCUSSION POLL REVIEW

Topic

Taylor Swift

Artist, Album, or Song

Title

Title

Discussion Content

Create Post

Step 5a. Fill out the body of your post, this can be up to 2000 characters long.

New Post

DISCUSSION POLL REVIEW

Topic
Taylor Swift

Artist, Album, or Song
Title
Title

Discussion Content
Content Content Content Content Content Content Content Content
Content

Create Post

Step 6a. Click the “Create Post” button to create your post.

New Post

DISCUSSION POLL REVIEW

Topic
Taylor Swift

Artist, Album, or Song
Title
Title

Discussion Content
Content Content Content Content Content Content Content Content
Content

Create Post

Poll Posts:

Step 3b. Fill out the topic by typing in the name of an artist, album, or song. If the entry exists on the database then the user will be given topics that match what they typed and they can select a topic.

New Post

DISCUSSION

POLL

REVIEW

Topic

news of the w

News of the World (Album)

News of the World (Song)

News of the World (Song)

☒ Poll Option 1

☒ Poll Option 2

☒ Poll Option 3

Remove Option

Add Option

Create Post

Step 4b. Fill out a title for your post, this can be up to 128 characters long

New Post

DISCUSSION

POLL

REVIEW

Topic

News of the World

Artist, Album, or Song

Title

Title

☒ Poll Option 1

☒ Poll Option 2

☒ Poll Option 3

Remove Option

Add Option

Create Post

Step 5b. Fill out the poll options for the post which must be less than 100 characters, the user can use the “Add Option” and “Remove Option” buttons to add or remove poll options, to a minimum of 2 and a maximum of 8.

New Post

DISCUSSION

POLL

REVIEW

Topic

News of the World

Artist, Album, or Song

Title

☒

Poll Option 1

option 1

☒

Poll Option 2

option 2

☒

Poll Option 3

option 3

Remove Option

Add Option

Create Post

Step 6b. Click the “Create Post” button to create your post.

New Post

DISCUSSION

POLL

REVIEW

Topic

News of the World

Artist, Album, or Song

Title

☒

Poll Option 1

option 1

☒

Poll Option 2

option 2

☒

Poll Option 3

option 3

Remove Option

Add Option

Create Post

Review Posts:

Step 3c. Fill out the topic by typing in the name of an artist, album, or song. If the entry exists on the database then the user will be given topics that match what they typed and they can select a topic.

New Post

DISCUSSION

POLL

REVIEW

Topic

emerson, lake

1

Emerson, Lake & Palmer (Artist)

Emerson, Lake & Powell (Artist)

Emerson, Lake & Palmer (Album)

Emerson, Lake & Powell (Album)

Create Post

Step 4c. Fill out a title for your post, this can be up to 128 characters long

New Post

DISCUSSION

POLL

REVIEW

Topic

Emerson, Lake & Palmer

Artist, Album, or Song

Title

★★★★☆

Review

Create Post

Step 5c. Fill out the body of your post, this can be up to 2000 characters long, additionally, click on the stars to set a rating for your review.

New Post ×

DISCUSSION POLL **REVIEW**

Topic
Emerson, Lake & Palmer G

Artist, Album, or Song

Title
Title ★★★★★

Review
Content!!!! G

Create Post

Step 6c. Click the “Create Post” button to create your post.

New Post ×

DISCUSSION POLL **REVIEW**

Topic
Emerson, Lake & Palmer G

Artist, Album, or Song

Title
Title ★★★★★

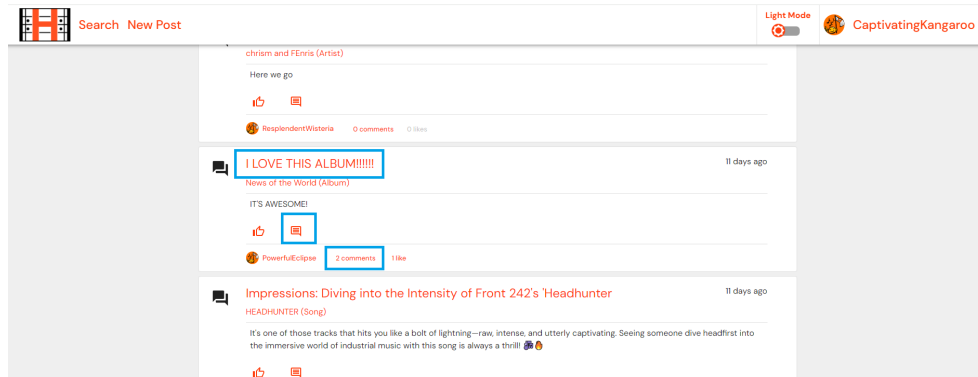
Review
Content!!!! G

Create Post

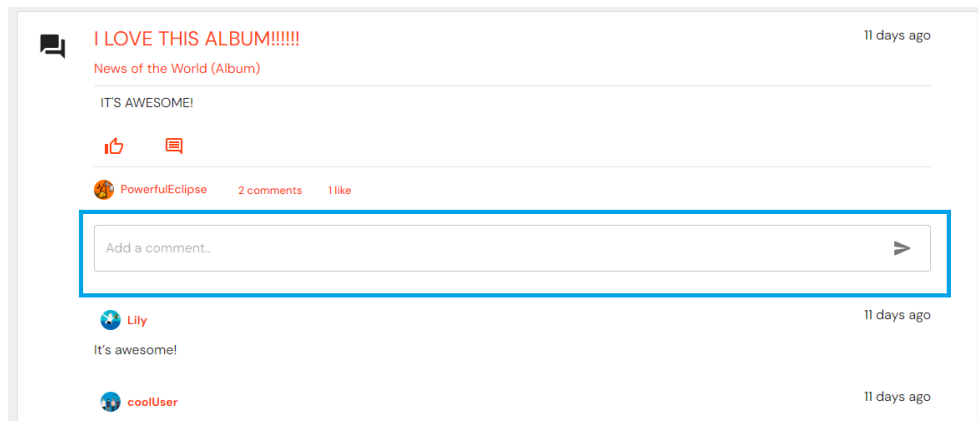
Interact With posts

Comment:

Step 1. User must be logged in to comment on posts. Find a post you would like to comment on and either click on the title of the post, or on the comment button.

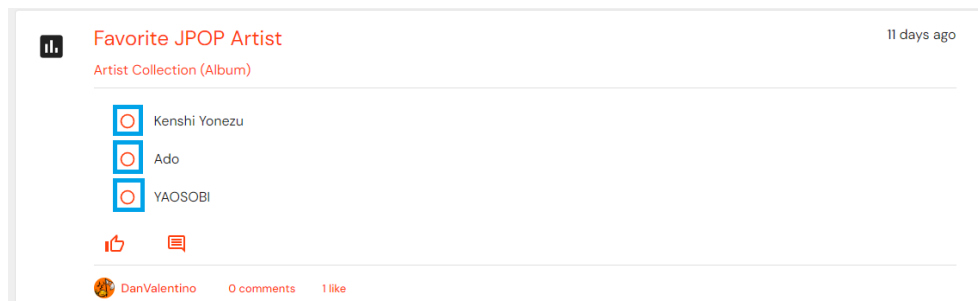


Step 2. A comment box will appear. Type your comment in and click on the button on the right to send the comment.



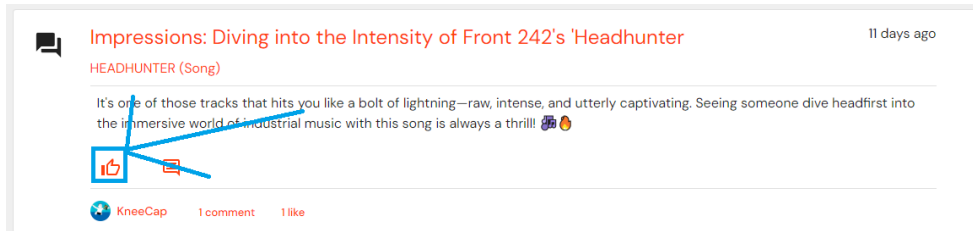
Vote:

Step 1. User must be logged in to vote on polls. Find a poll you would like to vote on and click one of the buttons to the left of the options.



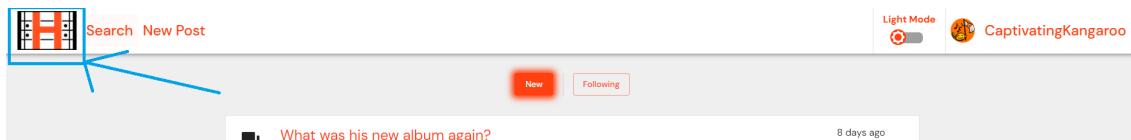
Like:

Step 1. User must be logged in to like posts. Find a post you would wish to like and press the like button.



Endless Feed of New Posts:

Step 1. Go to the home page by pressing the Harmony icon on the left side of the app bar

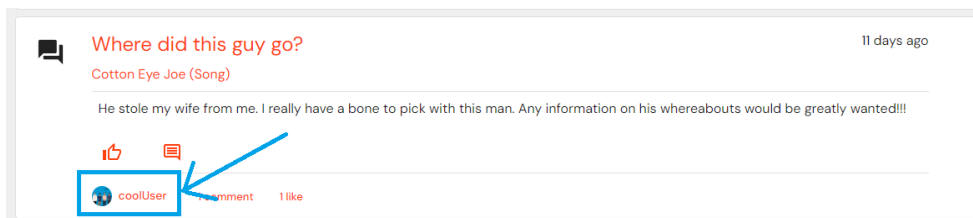


Step 2. The feed is on new by default, simply scroll down and posts will be loaded in as the user scrolls down.

Interact With Other Users

View Another User's Information, Posts, and Comments:

Step 1. Find a user to view and click on their username or search for them to go to their profile.



Step 2. The user can see the other user's name, bio, and posts.

The image shows a user profile for 'coolUser' with a bio 'cool cool' and 5 followers. The 'Content' tab is active, displaying two posts. The first post, 'What was your favorite lyric in this song?' for 'Message in a Bottle (Song)', has four identical options: 'Sending out an SOS'. The second post, 'Where did this guy go?' for 'Cotton Eye Joe (Song)', has a comment: 'He stole my wife from me. I really have a bone to pick with this man. Any information on his whereabouts would be greatly wanted!!!'. The 'Info' tab shows a welcome message: 'Welcome to the cool zone. Follow me for free V-Bucks.'.

coolUser
cool cool
5 Followers

Content

POSTS COMMENTS

What was your favorite lyric in this song? 11 days ago
Message in a Bottle (Song)

- ☐ Sending out an SOS
- ☐ Sending out an SOS
- ☐ Sending out an SOS
- ☐ Sending out an SOS

coolUser 0 comments 1 like

Where did this guy go? 11 days ago
Cotton Eye Joe (Song)

He stole my wife from me. I really have a bone to pick with this man. Any information on his whereabouts would be greatly wanted!!!

Info
Welcome to the cool zone. Follow me for free V-Bucks.

Step 3. Click on the “Comments” tab to view all of the comments the user has made.

The image shows the same user profile for 'coolUser', but the 'COMMENTS' tab is now active. It displays two comments: 'coolUser on I LOVE THIS ALBUM!!!!!!' with the text 'I agree, truly one of the best', and 'coolUser on Where did this guy go?' with the text 'Please anyone have info? Im really desperate'. The 'Info' tab remains visible on the right.

coolUser
cool cool
5 Followers

Content

POSTS COMMENTS

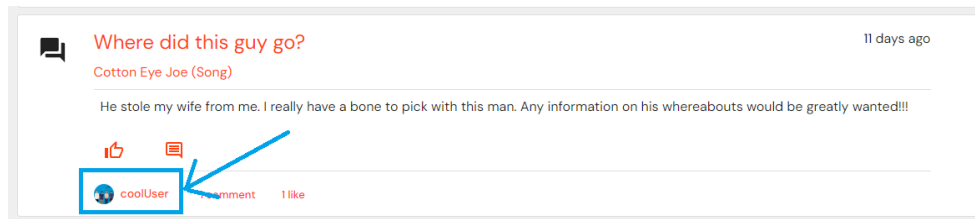
coolUser on I LOVE THIS ALBUM!!!!!! 11 days ago
I agree, truly one of the best

coolUser on Where did this guy go? 11 days ago
Please anyone have info? Im really desperate

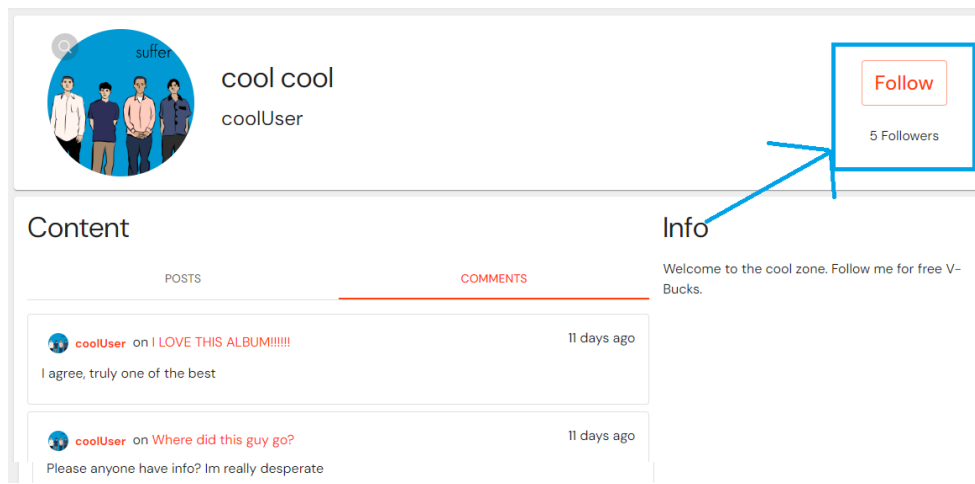
Info
Welcome to the cool zone. Follow me for free V-Bucks.

Follow Users and See their Posts on a Following Feed:

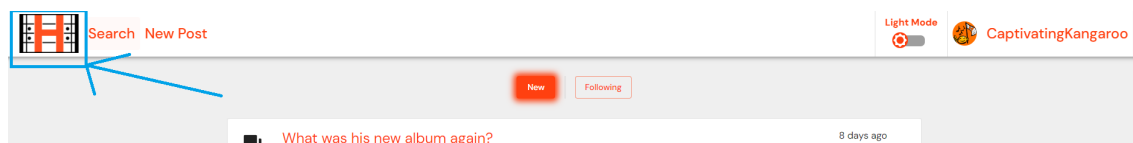
Step 1. The user must be logged in to follow a user. Find a user to view and click on their username or search for them to go to their profile.



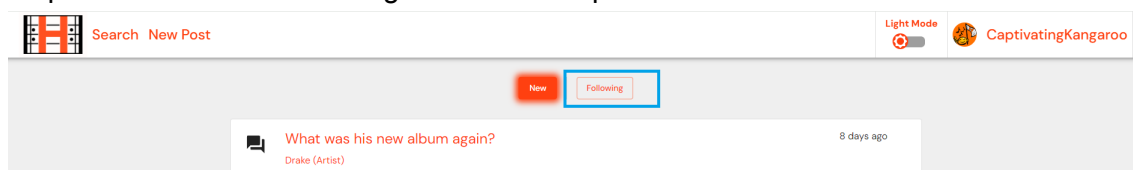
Step 2. Click on the “Follow” button on the top right of the page.



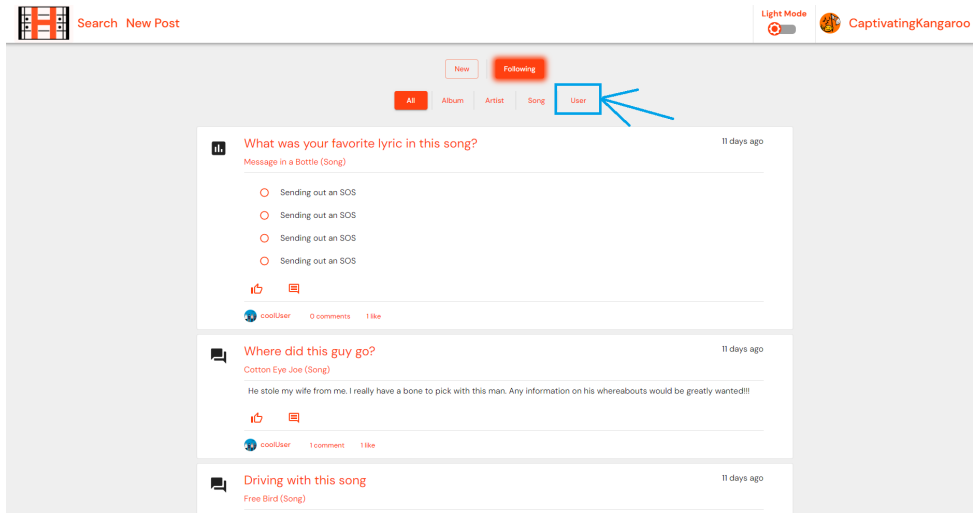
Step 3. Return to the home page by clicking the Harmony icon on the left side of the app bar.



Step 4. Click on the “Following” button to swap feeds.



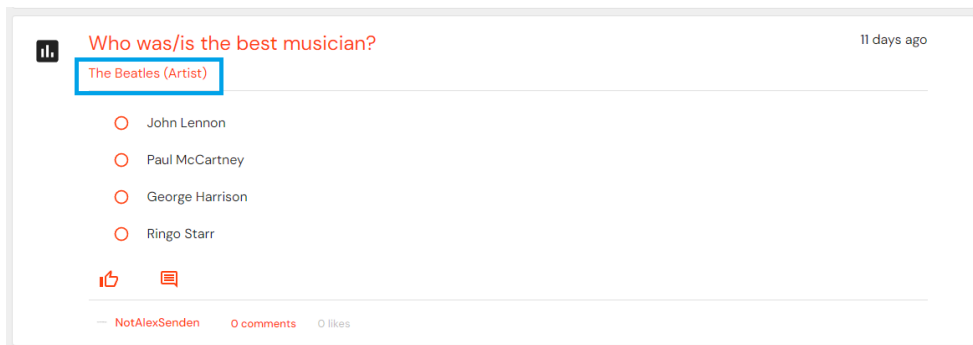
Step 5. Here, the user can see all the posts related to things they are following. Additionally, the user can see posts only related to followed users by click on the “User” button.



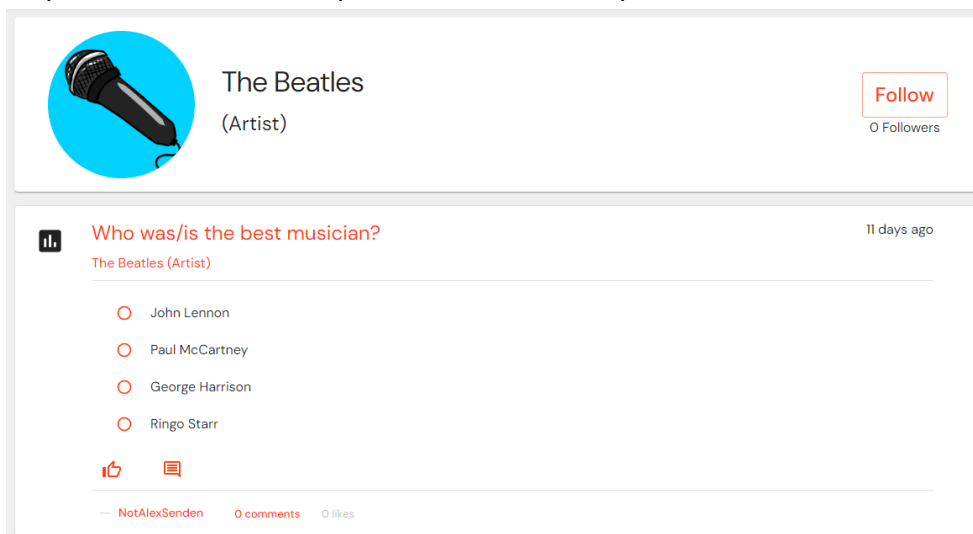
Interact With Artists/Albums/Songs

View Posts Relating to Specific Artists/Albums/Songs:

Step 1. Find a topic to view and click on the name or search for it to go to its page.

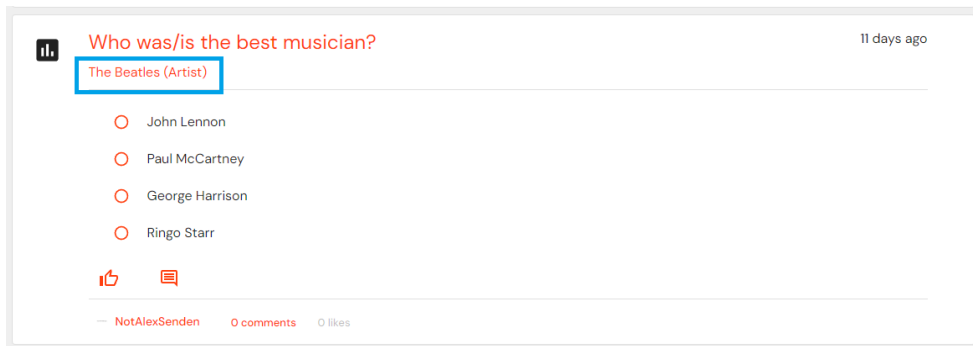


Step 2. The user can see posts related to the topic towards the bottom of the page.

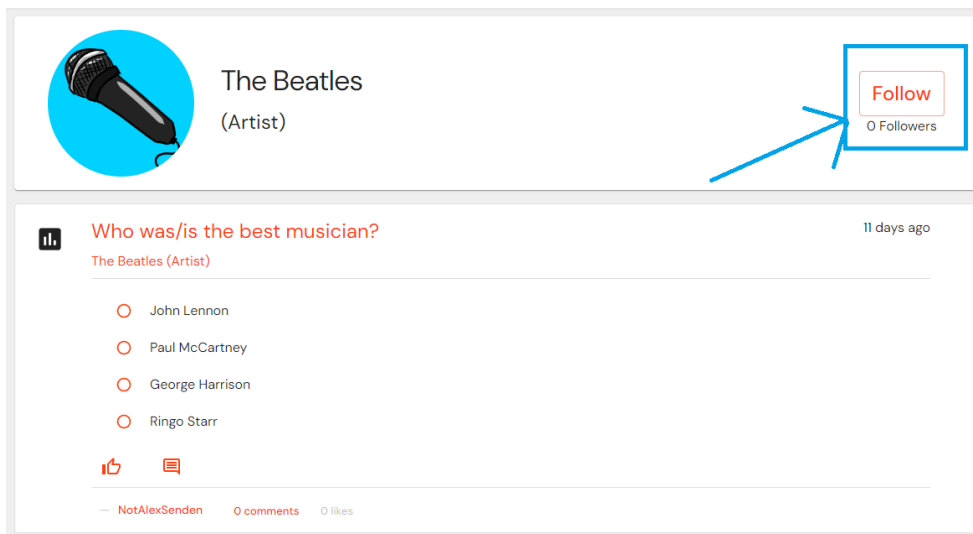


Follow Artists/Albums/Songs and See Related Posts on a Following Feed:

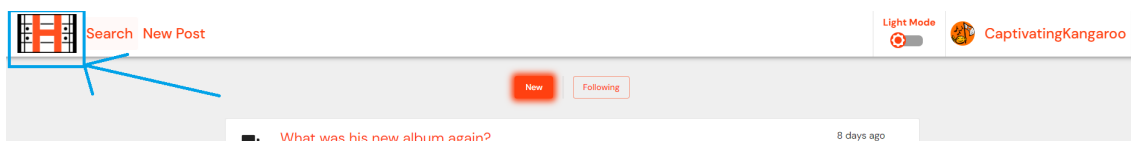
Step 1. User must be logged in to follow artists/albums/songs. Find a topic to view and click on the name or search for it to go to its page.



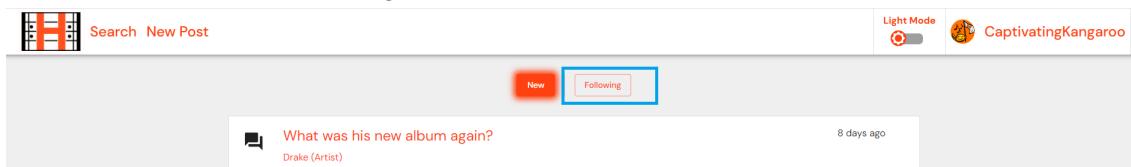
Step 2. Click on the “Follow” button on the top right of the page.



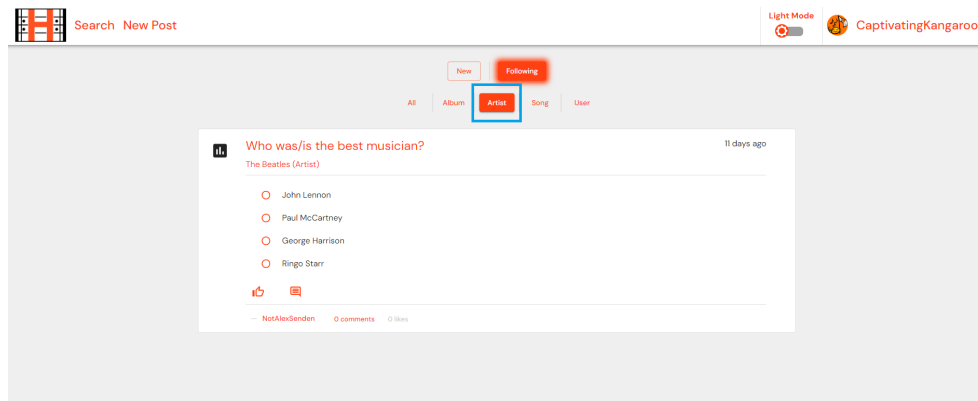
Step 3. Return to the home page by clicking the Harmony icon on the left side of the app bar.



Step 4. Click on the “Following” button to swap feeds.



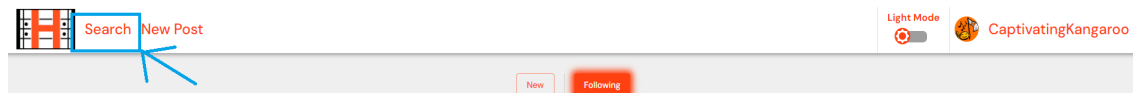
Step 5. Here, the user can see all the posts related to things they are following. Additionally, the user can see posts only related to followed topics by click on the “Album”, “Artist”, or “Song” button, the example below uses the “Artist” option.



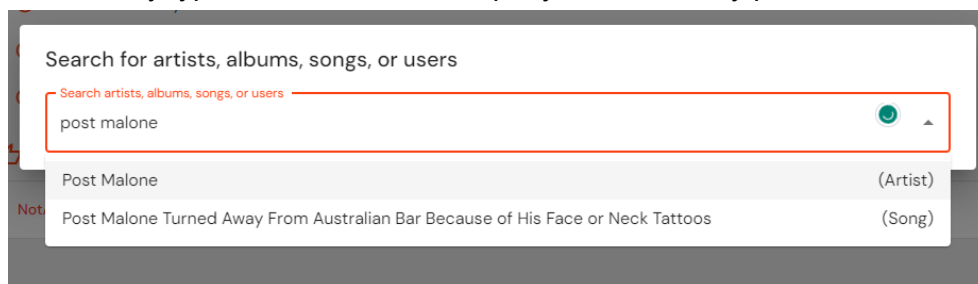
Search

Search for Artists/Albums/Songs/Users:

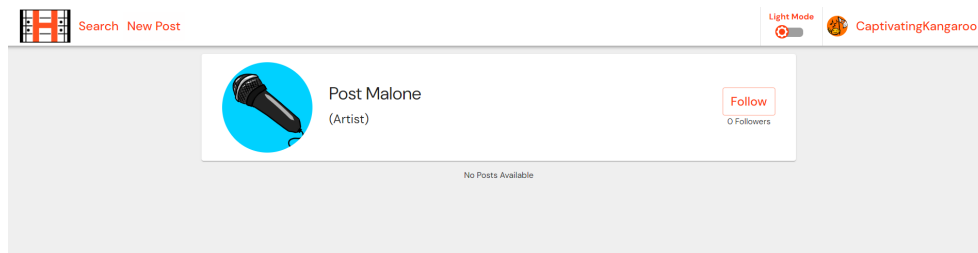
Step 1. Click on the “Search” button on the left side of the app bar.



Step 2. Begin typing in the name of an artist, album, song, or user. The user does not need to fully type in the name as the query will return any partial matches.



Step 3. Click on one of the options to go to their profile page in the case of users, and topic pages in the case of artists/albums/songs.



Overall Architecture and Design

[\[Link to architecture\]](#)

The architecture diagram can be found on page 11. You may need to scroll down, and press the load more button a few times

Infrastructure

FE:

[MUI:](#)

We decided to use MUI because it is very well documented, and saves a lot of time with their pre-made components. MUI stood out over other UI libraries since the components it provides are very flexible and easy to configure. On top of that, MUI is owned by Google, which further increases the developmental backing to this UI library

[Next:](#)

We decided to use NextJs because it allows us to use React with further optimizations. Next ships with React, however, will compile down the React pages, and cache them. We chose Next over just React since we wanted to optimize as much as possible for our deployment to Azure servers.

[Typescript:](#)

We decided to use Typescript because it lets us catch simple errors/bugs during the Typescript compile step. Typescript was chosen over just Javascript because Typescript will transpile down to Javascript while catching more errors. Further, Typescript is developed by Microsoft, leading to a strong developmental backing, and well-documented behaviour.

[Axios:](#)

We decided to use Axios because it simplifies HTTP requests from the front end to the back end. Axios works well with our usage of Next, with its React state management. Overall, Axios lets us create HTTP requests with well-documented behaviour.

[GTS:](#)

We decided to use the Google Typescript Format for Harmony because it makes the entire project have one naming scheme. We chose GTS over other naming conventions because GTS integrates well with our linter, which enforces the coding standard. Since GTS is backed by Google, the format is well-documented and generally easy to pick up.

[UUID:](#)

We needed a way to generate UUIDs, however, a custom implementation of UUIDs gets messy fast. We picked this UUID library specifically because it supports many UUID versions. Further, this UUID library sees over 90 million weekly downloads, showing its usefulness.

API:

Cookie-Parser:

We use cookie parsing middleware to isolate cookies out of the headers for HTTP requests. This allows us to use cookies at the very start of the request route since we do not need to create a custom implementation of grabbing cookies from the HTTP request. On top of that, this middleware works perfectly with Express, which further elevates its strength in our API

Body-Parser:

We use body parsing middleware to isolate the body out of the HTTP requests. This allows us to use the body at the very start of the request route since we do not need to create a custom implementation of grabbing the body from the HTTP request. On top of that, this middleware works perfectly with Express, which further elevates its strength in our API

CORS:

We use a CORS middleware to prevent users from hitting our API from websites that are not the defined Harmony front-end. We needed a CORS middleware since Harmony has an SSL certificate for HTTPS, which needs a CORS policy to leverage the use of cookies. This middleware also adds a level of security to Harmony, that makes sure that users are using the official front-end.

Dotenv:

We use dotenv for environment variables. We leverage the fact that dotenv allows us to use secrets in the context of a production build, and .env files in the context of a developer running the project locally. Overall, the use of dotenv reinforces the coding standard of not placing critical API keys in your code.

Express:

We needed a way to handle our HTTP requests in the back-end, so we went with Express. Express is very well documented, which allows us to understand its behavior very well. We chose Express over other routing frameworks, since Express works very well with middleware that overall simplifies our routes.

Jest:

We needed a way to make sure that Harmony is rigid, so we used Jest to test it. Jest provides a detailed report on our test suits, providing file, statement, and branch coverage. We chose Jest over other testing frameworks since Jest works very well with our chosen tech stack.

Supertest:

We needed a way to pass HTTP requests to our tests, so we used Supertest for that. Supertest gives us the ability to start our unit tests from the very start of the seam in the API, leading to more reliable coverage. Supertest works very well with Jest, which is why we chose it over other similar libraries.

ESLint:

Everyone has their own style of coding, however, ESLint enforces the GTS coding style. We utilized ESLint in our PRs and prevented any code that did not follow the coding style from being merged into the main branch. We chose ESLint over other linters because it is easy to set up, and will give understandable messages concerning improper formatting.

Prettier:

With our linter, we needed a formatter, which we use Prettier for. Prettier works hand-in-hand with ESLint to re-format code deemed to not be following the coding style. On command Prettier will automatically re-format the code in a given file, saving time for us by making the code more readable, and consistent.

Prisma:

The API needed a way to interact with the database, so we decided that

we would use the Prisma ORM. Prisma provides an abstraction to the database, whilst providing typed database columns, which integrates well with Typescript. We chose Prisma over other ORMs since Prisma works very well with PostgreSQL

DB:

[PostgreSQL:](#)

Harmony needed a relational database to effectively create our complex user-to-user interactions. PostgreSQL stuck out from other SQL dialects since it is a tried, and true object-relational database system. Further, we knew that we would be utilizing an ORM like Prisma, and Postgres seamlessly works with ORMs.

Load Testing:

[Apache JMeter:](#)

Harmony uses Apache JMeter for our load testing because it is very well documented and easy to use. We were also given a tutorial on how to create load tests using Apache JMeter in class, so the team was familiar with the software. The familiarity and ease of use made Apache JMeter the easy choice for load testing software.

Name Conventions

List your naming conventions or just provide a link to the standard ones used online.

Harmony follows a slightly modified version of the [Google Typescript Style \(GTS\)](#) for file naming and code style.

The specific stylistic modifications from GTS are a maximum line width of 80 characters, no semicolons, and bracket spacing (for example, { this } and not {this}).

Additionally, Harmony uses kebab-case for frontend components.

The styling integrates with the project's linter ([ESLint](#)) and code formatter ([Prettier](#)).

Most Important Code Files

File With GitHub Link	Purpose
harmony-fe/src/components/post-feed/post-feed.tsx	This file handles the fetching of posts from the backend and the displaying of them in an endless feed for all of the different feeds on the website.
harmony-fe/src/components/post/comment-section.tsx	This file fetches the comments for a post, and handles the logic for the client-side rendering of new comments without refetching from the backend.
harmony-fe/src/contexts/userContext.tsx	This file contains the logic for the global user state in the frontend. This allows for the frontend to always be aware of the login status and the current user.
harmony-api/src/services/userService.ts	This file handles all of the business logic for actions relating to user accounts, such as registration and session management.

[harmony-api/src/repos/postRepo.ts](https://github.com/alexsenden/harmony/blob/main/src/repos/postRepo.ts)

This file handles all of the interactions with the database regarding posts, including the paginated fetching of all of the post feeds and their filters (where applicable).

Continuous Integration and Deployment (CI/CD)

Continuous Integration

The Harmony CI pipeline executes on every pull request to the main branch (and re-run on pushes to branches with open pull requests). The CI is split into 2 main sections: The frontend CI and the backend CI. These sections are both implemented using GitHub actions. All jobs in both workflows must be successful before a pull request can be merged.

Frontend CI:

<https://github.com/alexsenden/harmony/blob/main/.github/workflows/fe-ci.yaml>

The frontend CI has two jobs.

- 1) Build the project. This ensures that there are no compile-time errors with the code.
- 2) Test and lint the project. The frontend has no tests, but if unit tests were added, they would be automatically tested by the CI. Additionally, this job runs the linter on the frontend codebase. If the linter raises any warnings or errors, or if any of the tests fail, then this job will fail.

Backend CI:

<https://github.com/alexsenden/harmony/blob/main/.github/workflows/api-ci.yaml>

The backend CI has 4 jobs:

- 1) Build the project. This ensures that there are no compile-time errors with the code.
- 2) Unit tests. This job runs all of the backend's unit tests. If any of the tests fail, this job fails.
- 3) Integration tests. This job sets up a new remote database and runs all of the backend's integration tests against this remote database. Once the tests are completed, this database is destroyed. If any of the integration tests fail, this job fails.
- 4) Lint. This job runs the linter on the backend codebase. If the linter raises any warnings or errors, then this job fails.

Finally, the CI runs a SonarCloud security scan via an integration with SonarCloud (not GitHub actions, although it does appear in the "Checks" section on GitHub). The changes must pass the quality gate seen below before it can be merged.

Metric	Operator	Value
Coverage	is less than	80.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

- 1) Describe your CI/CD environment and the clickable link to your CI/CD pipeline. For instance, if you use GitHub Action, provide the link to the workflow; if you use Jenkins, provide the link to the pipeline file.
- 2) Snapshots of the CI/CD execution. Provide one for CI and one for CD to demo your have successfully set up the environment.

Continuous Deployment

The harmony CD pipeline executes on merged pull requests into the main branch. It is all connected in two scripts that build docker containers and push them to DockerHub and one script that does the rest.

The two scripts that build docker containers are separated simply to separate the frontend docker build from the backend.

Frontend docker build:

<https://github.com/alexsenden/harmony/actions/runs/7226808429>

Backend docker build:

<https://github.com/alexsenden/harmony/actions/runs/7226808417>

Steps:

- Connect to registry
- Build container from the repo
- Push the container to the repo

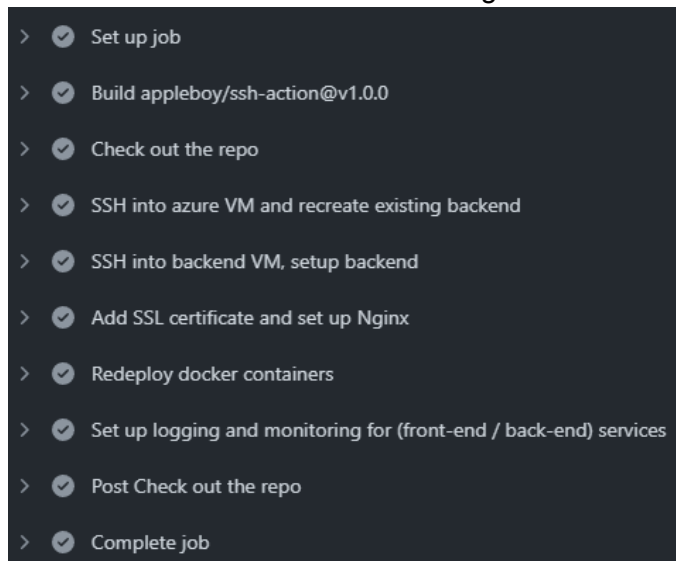
Everything is done with the help of a small Github Actions library that helps simplify this process.

The one script that does the rest is broken down into a few meaningful parts.

<https://github.com/alexsenden/harmony/actions/runs/7226808428>

Script Steps:

- SSH into Azure VM and recreate existing backend
 - Connect to permanent VM that has access to a few VM commands and run:
 - Delete existing VM
 - Create new VM
- SSH into backend VM, setup backend
 - Install necessary libraries on VM and update Ubuntu libraries to the latest version
- Add SSL certificate and set up Nginx
 - Download certificate for secret storage and set it up on Nginx proxy server. It acts as middleware to our docker services
- Redeploy docker containers
 - Pull docker container of docker registry and run them
 - They were built in the previously mentioned two scripts
- Set up logging and monitoring for (front-end / back-end) services
 - Install necessary agent and configurations to collect logs, and then push them to Grafana storage



Testing

Link to Testing Plan

<https://github.com/alexsenden/harmony/blob/main/documentation/Harmony%20-%20Test%20Plan.pdf>

Unit/Integration/Acceptance Tests

Unit tests can be run locally using the command `npm run unit-test`, and are also run in CI. The unit tests have 100% branch coverage in the backend. There are 204 unit tests.

Integration tests can be run locally using the command `npm run integration-test` with a local Harmony database instance running, and are also run in CI. There are 51 integration tests.

10 Most Important Unit Tests

Test File	What It's Testing
harmony-api/ __test__ /services/postService.test.ts (validates and returns discussion posts)	This tests the happy path for the validation logic for discussion posts. If this fails, then no new discussion posts can be created.
harmony-api/ __test__ /services/postService.test.ts (validates and returns poll posts)	This tests the happy path for the validation logic for poll posts. If this fails, then no new poll posts can be created.
harmony-api/ __test__ /services/postService.test.ts (validates and returns review posts)	This tests the happy path for the validation logic for review posts. If this fails, then no new review posts can be created.
harmony-api/ __test__ /repos/postRepo.test.ts (mapPrismaPostToPost. Returns empty string or undefined for null fields)	This test ensures that the logic mapping the Prisma ORM Post model to Harmony's internal domain Post model works properly.
harmony-api/ __test__ /controllers/post/following/followingController.test.ts (getFollowingFeed. Throws an error when the feed type is invalid)	This test ensures that only valid following feed filters can be passed when fetching a user's following feed.
harmony-api/ __test__ /services/userService.test.ts (validateUserRegistrationAdvanced. throws an error if the username fails regex)	This test ensures that only valid usernames that pass the username regex can be registered.
harmony-api/ __test__ /services/userService.test.ts (validateUserRegistrationAdvanced. throws an error if the password fails regex)	This test ensures that only valid passwords that pass the password regex can be registered.
harmony-api/ __test__ /services/topicService.test.ts (getTopicByPartialName. returns empty array)	This tests one of the most frequently called functions to ensure that it returns nothing early if no data is passed.
harmony-api/ __test__ /services/topicService.test.ts (getTopicOrUserByPartialName. returns empty array)	This tests one of the most frequently called functions to ensure that it returns nothing early if no data is passed.
harmony-api/ __test__ /controllers/user/userController.test.ts (Register Controller. should register a user)	This ensures that the registration controller correctly maps user data to the internal domain object and sets the user's session

	cookie in the response.
--	-------------------------

5 Most Important Integration Tests

Test File	What It's Testing
harmony-api/ __test__ /repos/postRepo.integration.ts (Should create a post for the user)	Tests the seam between the backend and the database for creating new posts.
harmony-api/ __test__ /repos/postRepo.integration.ts (Should get trending posts)	Tests the seam between the backend and the database for fetching the new/trending feed (the main one that you see when you view the homepage).
harmony-api/ __test__ /repos/artistRepo.integration.ts (Responds with an array of artists whose names are prefixed by a certain string)	Tests the seam between the backend and the database for searching for artists by name (the most common search, used in typeahead).
harmony-api/ __test__ /repos/userCookieRepo.integration.ts (should assign a cookie to the user)	Tests the seam between the backend and the database for retrieving a user's session.
harmony-api/ __test__ /repos/commentRepo.integration.ts (creates a comment in the DB and responds with the comment object)	Tests the seam between the backend and the database for creating new comments on posts.

5 Most Important Acceptance Tests

GitHub Link to User Story (Issue) With Acceptance Criteria in Description	User Story Description
https://github.com/alexsenden/harmony/issues/10	As a new user, I want to create an account so that I can interact with post.
https://github.com/alexsenden/harmony/issues/13	As a logged-in user, I want to create discussions so that I can interact with other users and see their opinions.
https://github.com/alexsenden/harmony/issues/18	As a logged-in user, I want to comment on posts so that I can share my thoughts.
https://github.com/alexsenden/harmony/issues/16	As a user, I want to view other user's profile information so that I can understand who the user is.
https://github.com/alexsenden/harmony/issues/26	As a logged-in user I want to follow my favourite artists/albums/songs so that I can participate in recent discussion.

Regression Testing

Regression testing is run on every pull request (and re-run on pushes to branches with open pull requests) via GitHub actions using Jest. The regression tests must pass before the change can be merged. The regression tests consist of the full suite of unit and integration tests for Harmony.

[Lines 23-71 of `api-ci.yaml`](#) execute the regression tests. The unit tests are run against the backend, and the integration tests are run against the backend and a fresh remote database.

Unit Test Execution Snapshots:

```
131 Test Suites: 41 passed, 41 total
132 Tests:       204 passed, 204 total
133 Snapshots:   0 total
134 Time:        15.691 s
135 Ran all test suites.
```

```
49 -----|-----|-----|-----|-----|-----
50 File           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
51 -----|-----|-----|-----|-----|-----
52 All files      |    100 |     100 |     100 |     100 |
```

Integration Test Execution Snapshot:

```
616 Test Suites: 10 passed, 10 total
617 Tests:       51 passed, 51 total
618 Snapshots:   0 total
619 Time:        3.855 s
620 Ran all test suites.
```

A full execution of a recent regression test execution can be found at:

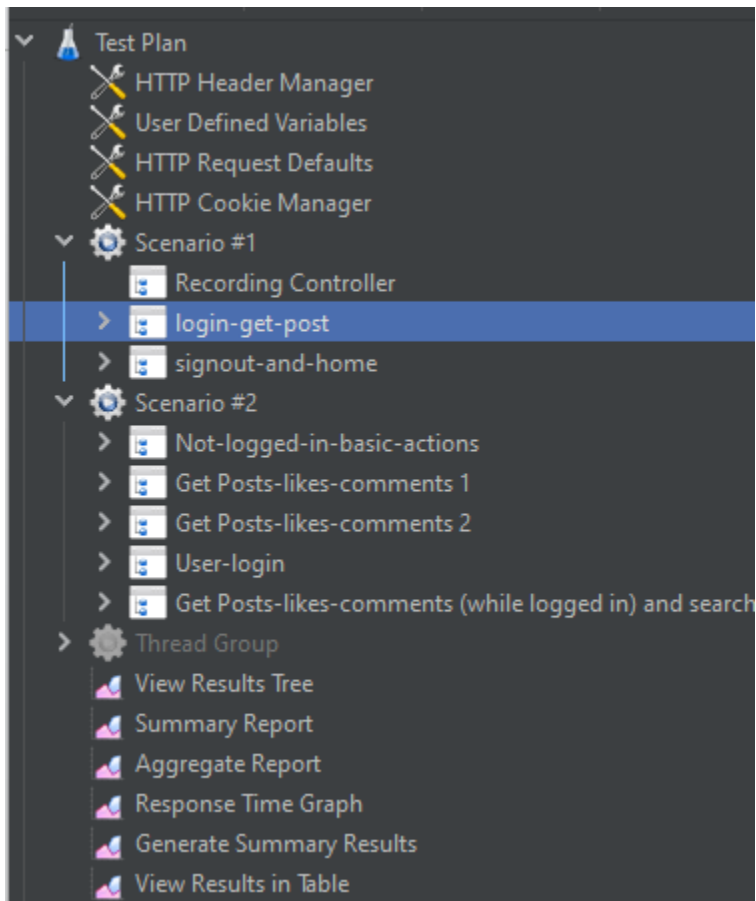
<https://github.com/alexsenden/harmony/pull/156/checks>

Load Testing

For load testing, we used the common tool - Apache JMeter. It is well-documented and easy to use.

Our test cases were mostly Get requests, but also some Post requests that are not making changes to the database which can cause race conditions (for example, the Login feature, which adds a cookie and will not cause an issue).

We grouped out test cases in user scenarios, and they were extracted using the JMeter tool for the following steps. Then, we adjusted these test cases to fill them with necessary or missing data.



For example, to keep the cookie I needed the following Beanshell postprocessor for each user thread:

```

1 import java.util.regex.Matcher;
2 import java.util.regex.Pattern;
3
4 String responseHeaders = prev.getResponseHeaders();
5
6 String pattern = "UserCookie=[\\w-]+";
7 Pattern r = Pattern.compile(pattern);
8 Matcher m = r.matcher(responseHeaders);
9
10 log.info("Headers: " + responseHeaders);
11
12 if (m.find()) {
13     String extractedValue = m.group(1);
14     props.put("UserCookie" + ctx.getThreadNum(), extractedValue);
15     log.info("UserCookie Value: thread " + ctx.getThreadNum() + " : " + props.get("UserCookie" + ctx.getThreadNum()));
16 }

```

Here is a link to the output of test results of 30 users over 60 seconds:

<https://drive.google.com/file/d/162C7IzLvZ8RD9jLO4wIMxK1aCWJrhuKO/view?usp=sharing>

We did not have any bottlenecks besides obvious performance drops when the amount of users exceeds approximately 500 with 15 requests per user per second, when the server might stop responding to additional users.

Link to .jmx file on our github repo:

https://github.com/alexsenden/harmony/blob/main/load-testing/Main_Test_Plan.jmx

Security Analysis

Harmony uses SonarCloud for code scanning. A security scan is executed on every pull request (and re-run on pushes to branches with open pull requests). The scan must pass the quality gate before the changes can be merged. The security scan is run via an integration with SonarCloud, and therefore does not require any workflow file in the Harmony repository.

Harmony is written in TypeScript, which is supported by SonarCloud.

- 1) [React Hook "useEffect" is called conditionally. React Hooks must be called in the exact same order in every component render.](#)
 - a) This issue is not a false positive and is something that should be fixed. It is true that React hooks should not be conditionally called. In this case has been working because the condition should always be true, however it should still be changed regardless.
- 2) [Expected an assignment or function call and instead saw an expression.](#)
 - a) This is another real problem that was caught by SonarCloud that should be fixed by the Harmony team. The result of the expression should be returned.
- 3) [Replace this "String" wrapper object with primitive type "string".](#)
 - a) Although there is no effect on the performance, this is a consistency issue that should be fixed.
- 4) [Non-interactive elements should not be assigned mouse or keyboard event listeners.](#)
 - a) Again, this issue has no behavioural impact, however for clarity should be wrapped in some sort of button component.
- 5) [useState call is not destructured into value + setter pair.](#)
 - a) This is simply a variable naming convention issue, to follow the convention the setter function should likely be renamed to `setIsLoading`. Again, no impact on the product, but a maintainability issue nonetheless.

The project scans can be found at:

https://sonarcloud.io/summary/new_code?id=alexsenden_harmony

Overall, the feedback generated by SonarCloud was mostly good. The bugs found by SonarCloud were often real issues that should be fixed by the team with few false positives, and the code smells were good indicators of areas in the code that could be cleaned up. The security hotspots were almost always false positives due to "hard coded credentials" in our test cases, which was entirely mock data. In conclusion, the SonarCloud security scans were effective at helping identify issues in our codebase.