UNSW
SYDNEY

School of Computer Science and Engineering
Faculty of Engineering
UNSW Sydney

UltraCast

BY

CAPSQUAD

Daniel Latimer	Connor O'Shea	Kevin Chan	Oliver Richards	Peter Kerr
z5115175	z5115177	z5113136	z5157383	z5115807

Submitted: November 16, 2020
Tutor: Tatjana Zrimec

Contents

1 Overview

1.1 Introduction

With over 100 million monthly listeners [15] and a steadily increasing user base, there is no doubt that podcasts are a greatly enriching source of information and entertainment for a large variety of individuals.

Although they are highly valuable, it can be difficult to find podcasts that are of interest to a particular user amidst the 1 million [15] that are already available. Thus, podcast streaming services (such as UltraCast) have been created, to provide a centralised place for exploring and discovering new podcasts that are valuable to the listener.

However, all of the web based podcast streaming services available lack many important features, and their interfaces leave much to be desired. For example, there is no streaming service that allows the user to bookmark certain parts of a podcast, nor take notes at certain timestamps. It is even difficult to find a service that allows the listener to change the playback speed of the podcast.

UltraCast combines all of the most important features together into a single package with a web-based podcast streaming service.

UltraCast differentiates itself from competitors by allowing users to:

- Follow friends to see what they have been listening to
- Create *streams* of podcasts to find interesting podcasts
- Create bookmarks inside podcast episodes
- Monitor episode and podcast play metrics

1.2 Project Requirements

The minimum project requirements from the specifications are:

- Listeners must be able to search for podcasts that interest them by keywords, resulting in a list of matching podcast titles, where the total number of subscriptions on the UltraCast platform (function described later) for each podcast is shown next to the title
- Listeners must be able to select a podcast show from returned search results to view its full details, including its title, description, any author details that exist, as well as a list of episodes for the show
- Listeners must be able to play a selected episode within a podcast show, and once that episode starts being played, the listener must be able to also clearly see this episode marked as *played*
- Listeners must be able to subscribe or unsubscribe from a podcast show Listeners must be able to see the latest episode available for each show that they subscribed to in a *Podcast Subscription Preview* panel
- Listeners must be notified by the platform when a new episode for a show they are subscribed appears

- Listeners must be able to see a history of the podcast episodes that they have played, sorted in order from most recently played to least recently played
- UltraCast must be able to recommend new podcast shows to a listener based on at least information about the podcast shows they are subscribed to, podcast episodes they have recently played, and their past podcast searches

The following additional requirements have also been implemented:

- Listeners should be able to follow their *friends* and view the podcasts that their *friends* have recently listened to
- Listeners should be able to create *streams* based off search queries that they can use to find interesting podcasts
- Listeners should be able to add bookmarks with a name and description to podcast episodes as they listen to them
- Content creators should be able to create and upload podcasts and podcast episodes
- Content creators should be able to monitor analytics of their uploaded podcasts related to their listeners

Requirements testing has been performed and the results can be found in Section A. UltraCast fulfils all requirements.

1.3 System Architecture

The high-level system architecture can be seen in Figure 1. The end users, podcast listeners and content creators, connect to the presentation layer which is powered by a ReactJS application. ReactJS was selected as the framework for the frontend application primarily due to it being a mature and world leading framework[9] as well as due to previous experience with the framework.

Flask, a python based micro-framework[2], was used for the web-server, due to its ease of use which allowed for rapid development. The React application communicates with Flask web-server through a GraphQL API: a scalable alternative to the popular REST API[3]. The Flask web-server also contains a recommendation service which drives recommendation functionality described in Sections 2.1.3 and 2.2. The Flask web-server uploads static files (episode audio and podcast covers) to a static file storage server while the urls for these static files along with all other data is stored in MongoDB. MongoDB, a NoSQL database, was selected to store meta-data due to its scalability[12]. MongoDB could not be hosted on CSE so was hosted on AWS EC2 instead, see Table 1 for reason. As a result the Flask web-server needed to be on the remote server as well to improve performance as described in Section 2.3. Algolia, a search engine service[1], was used to maximise search performance and reduce development time.

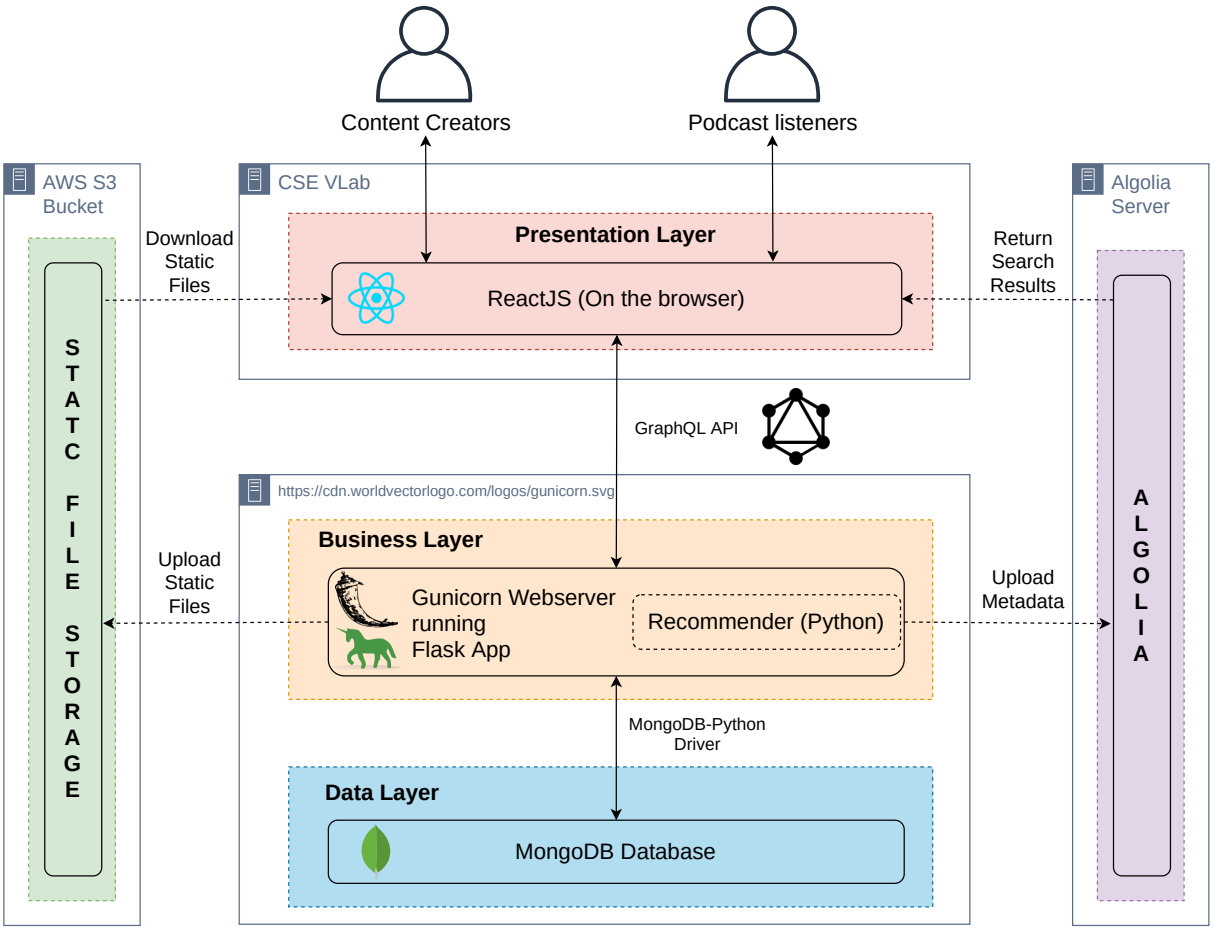

Figure 1: UltraCast System Architecture

1.3.1 Third Party Components

The third party components which had a notable impact on the functionality of UltraCast are shown in Table 1. This is not an exhaustive list of all the libraries and system libraries in UltraCast but rather a list of components which were notable and impacted functionality.

Table 1: Third Party Components

Name	Component Type	Reason For Use	License
AWS S3	Cloud: Static File Storage	Exceeded CSE server storage limits. > 80GB of test data	Custom[5]
AWS EC2	Cloud: Remote Server	MongoDB unsupported on Debian 6 (CSE VLab)	Custom[5]
Algolia	SaaS: Search Engine	Avoid building search engine from scratch	Custom[4]
React	Frontend Framework	Bootstrap functionality	MIT[10]
Flask	Micro Framework	Bootstrap functionality	BSD-3 Clause[13]
Graphene	Python GraphQL Library	Standard library	MIT[14]
Mongoengine	Python object data mapper for MongoDB	Standard library	MIT[11]
Pandas	Python data Library	Standard library	BSD-3 Clause[6]
NumPy	Python maths Library	Standard library	BSD-3 Clause[8]
Gunicorn	Python WSGI	Maximize server performance	MIT[7]

In terms of liscencing the 3-Clause BSD license was the most restrictive license involved. The custom liscencing, terms and conditions of other 3rd party components was largely disclaimers. Compared to the MIT license, the 3-Clause BSD license contains an extra non-attribution clause. As a result the either a modified MIT license or a 2-Clause BSD license would be best suitable (as the 2+ BSD license requires the copyright notice to be reproduced). To protect the authors of the UltraCast project a 3-Clause BSD license was selected for the project.

2 Functionalities and Implementation Challenges

2.1 Functionalities

The subsections below describe the functionalities of UltraCast broken down into 5 major functionality groupings. Each functionality is directly connected to a User Story, defined in the Proposal, as well as a project objective if it addresses it. The colour schema defined in Table 2 was used for functionalities in Tables 3 to 7.

Table 2: Functionality Table Colour Schema

Colour	Meaning
	Functionality relates to project objective
	Functionality not specified in project objective
	As above, but functionality is novel

2.1.1 Viewing and Searching

The viewing and searching functionalities are described in Table 3 below.

Table 3: Viewing and Searching Functionality Mapping

Story ID	Functionality	Project Objective
UL-2 UL-3	Use keywords to search for podcasts, return list of podcasts (See UL-4 for format) View the total number of subscribers for each podcast returned from a search	Listeners must be able to search for podcasts that interest them by keywords, resulting in a list of matching podcast titles, where the total number of subscriptions on the UltraCast platform (function described later) for each podcast is shown next to the title
UL-4	View the title, description, author details and list of episodes for a podcast	Listeners must be able to select a podcast show from returned search results to view its full details, including its title, description, any author details that exist, as well as a list of episodes for the show
UL-14	Login as specific user	-
UL-24	View a title, length, upload date for episodes	-
UL-29	Save search as a "Stream"	-
UL-41	Sign up as user	-

2.1.2 Playing Podcast Episode

The playing podcast episode functionalities are described in Table 4 below.

Table 4: Playing Episode Functionality Mapping

Story ID	Functionality	Project Objective
UL-5 UL-6	Play episodes Once episode starts being played it is marked as played	Listeners must be able to play a selected episode within a podcast show, and once that episode starts being played, the listener must be able to also clearly see this episode marked as "Played"

UL-18	Pause episode that is playing	-
UL-19	Adjust playback volume	-
UL-20	Skip to next episode, previous episode and start of current episode	-
UL-21	Jump to a point in an episode	-
UL-22	Adjust playback speed	-
UL-23	Auto-play episodes in a podcast (after added to playlist)	-
UL-26	"Bookmark" a point in an episode with a title and description	-

2.1.3 Recommendation and Following

The recommendation and following functionalities are described in Table 5 below.

Table 5: Recommendation and Following Functionality Mapping

Story ID	**Functionality**	**Project Objective**
UL-10	View episode history	Listeners must be able to see a history of the podcast episodes that they have played, sorted in order from most recently played to least recently played
UL-11	Episode history is sorted by most recent to least recent	
UL-12	Podcast recommendations are based on: Existing subscriptions recently played episodes and past searches	UltraCast must be able to recommend new podcast shows to a listener based on at least information about the podcast shows they are subscribed to, podcast episodes they have recently played, and their past podcast searches
UL-13	A "recommended" panel shows recommended podcasts	
UL-18	Follow users, view their listen history	-

2.1.4 Creator Mode

The creator functionalities are described in Table 6 below.

Table 6: Creator Mode Functionality Mapping

Story ID	**Functionality**	**Project Objective**
UL-15 UL-16	Create podcasts and episodes Delete podcasts and episodes	-
UL-17	Update podcasts and episodes	-
UL-27	Access to podcast and episode viewer metrics	-

2.1.5 Subscribe

The subscribe functionalities are described in Table 7 below.

Table 7: Subscribe Functionality Mapping

Story ID	Functionality	Project Objective
UL-7 UL-8	Subscribe to podcasts Unsubscribe to podcasts	Listeners must be able to see a history of the podcast episodes that they have played, sorted in order from most recently played to least recently played
UL-9	User receives notification for each new episode in a podcast they are subscribed to	Listeners must be notified by the platform when a new episode for a show they are subscribed appear
UL-30	The latest episode for each subscribed podcast is linked in the "Subscriptions" page	Listeners must be able to see the latest episode available for each show that they subscribed to in a "Podcast Subscription Preview" panel

2.2 System

2.2.1 GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling these queries. It provides a complete and understandable description of the data in your API, with a type system to ensure that apps can only make logical requests - providing clear and helpful errors when necessary. This works to improve the reliability of any solution built upon it.

GraphQL queries also return predictable results by allowing clients to define the structure of the data required; the queries supply exactly what a client asks for and nothing more. This ensures the speed and stability of the apps built using GraphQL, since the app itself controls the data it receives instead of the server. GraphQL also comes with a powerful developer GUI that allows you to visualise exactly what data your API has available. It also highlights potential errors in test queries and leverages code intelligence to make helpful suggestions on how to fix them.

GraphQL is naturally scalable, as any type or amount of data can be contained inside a single query. New queries can be developed on the fly and existing queries can be modified all without breaking existing functionality.

2.2.2 MongoDB

MongoDB is a document database designed to be used in the development of scalable applications using agile methodologies. MongoDB naturally supports rapid iterative development, which fits the nature of this project perfectly. It scales to high levels of read and write traffic and to a massive database size, and can easily evolve the type of deployment as necessary; removing roadblocks to development if the structure of our solution changes. As MongoDB relies on JSON files to store data, the structure of the information is under the control of the developer. Developers can adjust and reformat the database as the application evolves with ease - further supporting the RAD approach.

2.3 Implementation Challenges

Throughout the duration of the project, our team faced many implementation challenges. Outlined below is a series of the major challenges we encountered, how they impacted the project and how we overcame those challenges.

Technology stack and libraries UltraCast employed an unusual combination of technology as its core stack. MongoDB was used in the persistence layer, Flask as a web server framework, GraphQL (through the Graphene and Graphene-Mongo Python libraries) as the API layer and React for the frontend. Flask and React were chosen as some team members were familiar with it, while MongoDB and GraphQL were chosen due to their popularity and desire to learn them. Unfortunately, this decision process did not consider how these technologies would interact with each other. For example, GraphQL had been released in 2018, so there was not a lot of example applications that had it implemented in Flask. This problem had two different impacts: firstly, the team had to manage navigating through incomplete documentation; secondly, the team could find many examples to use while developing. As a result, additional research and prototyping had to be conducted in the first few sprints.

Halfway into the project, the team considered switching from Flask to Django, as the framework had better documentation and examples. After some research, it was decided that the change was not worth implementing, as most of the backend would have to be redone. Retrospectively, this decision seemed to be the right choice - the team became more familiar with the new technologies and less problems arose as time went on.

User Authentication Implementing user authentication for the backend was a non-trivial task because the Graphene and Graphene-Mongo libraries which are used for the API layer do not natively support this functionality. A major challenge in applying general purpose authentication libraries, for example flask-jwt[1], is that only one route is used for all API calls. Some of these API calls need to be authenticated e.g. deleting a podcast where others should not be e.g. signing up to the site. The Flask-GraphQL-Auth library[2] provides the required authentication methods, however, it is not actively maintained. After much research, user authentication was implemented using the flask-jwt-extended library[3]. This library allows authentication to be required on a per-function level, rather than for an entire route. Hence, certain GraphQL mutations and queries can be protected with user authentication where required. The frontend calls a sign in mutation which returns a Json Web Token (JWT). This mutation does not require authentication. The frontend then stores this JWT as a cookie and sends it in the header of any future GraphQL API requests. This token is then used to authenticate further queries.

Populating the Site To build a meaningful recommendation system, the website must have a reasonable amount of podcasts already uploaded to it. Since UltraCast has not been released, there are no users to generate this data. To allow for experimentation with different approaches to recommending podcasts to users, a podcast dataset was scraped from an online resource. It was difficult to find a suitable dataset that contained

[1] Available at https://github.com/mattupstate/flask-jwt

[2] Available at https://github.com/NovemberOscar/Flask-GraphQL-Auth

[3] Available at https://github.com/vimalloc/flask-jwt-extended

the required category, sub-category and keyword tags for podcasts that did not impose commercial obligations on UltraCast (due to terms of use of the dataset). A dataset which is an aggregation of public domain podcasts was found and scraped, providing over 200 podcasts and 2000 podcast episodes for the site.

Backend Business Layer Early in the project, the team did not believe that the backend needed to have a business layer implemented - the original specification was relatively simple, so no business rules had to be applied. However, due to the limitations of the technology stack chosen, it was later found to be necessary, as some work had been doubled up in the Flask server. For example, creating a podcast would occur twice - once between the frontend and backend (via GraphQL), and another between the backend and the database (via Graphene-Mongo). By implementing a business layer, all operations would go through a unified location, removing the need to double up on code.

Streamline Website Design While developing UltraCast, the design of the website became fractured. For example, it was possible to play a podcast in two different locations on the same page - creating confusion for developers. The reason this problem arose was because some team members had diverged from the original Figma designs, while others had not. The result was that some frontend work had been duplicated, wasting time in the sprint. To overcome this, the frontend team met and agreed to design future pages on Figma. By doing this, the design could be kept consistent and could be referred to when creating new pages.

Frontend State Management Originally, the frontend state was considered to be quite small - the user's information only had to be saved. As a result, the entire application state was saved into one variable in the top level React component, and was then passed around in the lower level components. As time went on, it became harder to manage state without affecting older features due to the single state variable. For example, the audio player had to keep track of the podcast queue and so the queue was saved into the state variable. Adding to the queue would trigger a re-render of the entire application and therefore break some older, more stable features. Ultimately, the state was refactored towards the end of the project into different variables, resolving this issue.

Resolving Nested Queries While testing the frontend, it was discovered that some backend GraphQL queries were taking upwards of one minute to return. The site was still responsive, however it took a long time for recommended podcasts to be displayed. Further investigation revealed that where nested references were used in the database models, and the GraphQL query involved dereferencing these references, the Graphene-Mongo library would perform one database operation per parent node. These database operations are performed sequentially. Since the MongoDB instance is hosted in the cloud, each database operation takes some number of milliseconds due to network latency. When a large number of parent nodes were fetched, this resulted in very slow queries. It was not feasible to modify the Graphene-Mongo library to issue less database operations. Hence, the decision was made to move the GraphQL API web server to the same cloud container as the MongoDB instance. This improved the time for some queries from over forty seconds to less than a second.

Database integrity During project development, the database lost integrity multiple times. Constant development changes to the database schema would result in records having missing or mismatched fields. This would result in the database becoming unstable, making it hard for the frontend to complete features that required a stable schema. To combat this, unit tests were created in the backend to ensure the stability of the database schema. However, if the schema and unit test changed, the issue would go unnoticed in the frontend. Towards the end of the project, a production and development copy of the database and backend server was created: this allowed the frontend to develop on the production system and made it more stable.

The 'jinke-music-player' library Initially, a small third-party library was used to play podcasts on the website - called 'jinke-music-player'. It was easy to setup and the team was able to complete multiple user stories at the same time, at once. However, the music player lacked some major functionality, namely, it could not handle fast seeking on the audio track (in other words, it could not jump forward to some section in the podcast that had not been loaded yet). The team had attempted to mount this functionality onto the library, but for technical reasons, this would only work roughly 80% of the time. Ultimately, the decision was made to remove the library completely, opting to use a less constrictive audio player.

3 User Manual

3.1 Software Setup Instructions

For the simple case where no API keys need to be changed, setting up and running UltraCast is as simple as running:

```
# cd to the root directory of the git repo
./start.sh
```

This script will:

- Create a python virtual environment (venv) for the backend
- Install all required python packages in the venv
- Install all npm packages that are required for the frontend
- Launch the backend web server (if the –local flag is used)
- Launch the frontend
- Open UltraCast in a web-browser (this may not work on VLab)

UltraCast can be run using either a local or remote GraphQL endpoint. The remote GraphQL endpoint is preferred due to lower latency (it must make successive MongoDB database operations which can be slow if network latency is high). It is highly recommended to run UltraCast in remote mode on UNSW CSE machines as the port used for the GraphQL endpoint is often already in use.

To run UltraCast using a local backend web server:

```
# cd to the root directory of the git repo
./start.sh --local
```

Once the web server is installed and running you will see a message like:

```
Serving frontend at http://localhost:43689/
```

You can then navigate to UltraCast in a web-browser at the link printed to the terminal e.g. `http://localhost:43689`. To avoid failing to launch because a port is already in use, the port which is used is decided at runtime. Ensure that you use the correct port.

3.2 Configuration

The following external services are used and their IP addresses and/or API keys will need to be set in configuration files:

- Algolia
- MongoDB Instance (hosted in a cloud container e.g. Amazon EC2)
- S3 Bucket
- Backend GraphQL endpoint (if not hosted on local machine)

Some of these variables need to be set for the frontend and some for the backend.

3.2.1 Backend Configuration

The backend is configured by using python files which set various configuration variables. These include options including:

- The IP address of the MongoDB instance
- The MongoDB database
- Flask secret keys (for encryption)
- Algolia API key and user

A full list of the variables that can be set is in `backend/config/default_settings.py`. Any variables that are not set are defaulted to the value in `backend/config/default_settings.py`. You can override these settings by writing a new python file and setting the environment variable `ULTRACAST_BACKEND_SETTINGS` to be the real path of this file. For example if the settings file is at `~/ultracast_settings.py`, you could do:

```
export ULTRACAST_BACKEND_SETTINGS=$(realpath ~/ultracast_settings.py)
bash backend/start.sh
```

3.2.2 Frontend Configuration

The frontend can be configured by editing the file `frontend/src/api/config.js` Here you can set options including:

- The backend GraphQL endpoint to use
- Algolia API key and user

3.3 Logging

When `./start.sh` or `backend/start_production.sh` are used to launch the backend, by default backend logs will be recorded to `backend/logs.txt`. This behaviour can be modified by changing the err-log option in `backend/backend_app.py`. When the backend is run in development mode, logs are printed to the terminal.

Frontend logs can be accessed via the web-browser console.

3.4 Integration Tests

Integration tests have been implemented to test the backend functionality. These start up a flask *test app* and then query it using GraphQL queries and mutations.
To run the integration tests:

```
cd backend
./run_tests.sh
```

If all tests are passed, something similar to the following will be printed:

```
Ran 32 tests in 121.359s

OK
```

3.5 Generating Podcast Data

UltraCast includes a script to generate podcast data based on this dataset of podcasts from 2007-2016: https://data.world/brandon-telle/podcasts-dataset. Note: The MongoDB database should already have this data unless you are creating a new MongoDB instance. Before running the script to generate data, ensure that you:

- Create or connect to your MongoDB instance (see Section 3.2.1)
- Download and unzip "episodes.csv" and "shows.csv" to backend/generate_data

Then run the following:

```
cd backend
source env/bin/activate
python -m generate_data.generate_data
```

3.6 Site Usage and Functionality Guide

3.6.1 Prerequisites

Please ensure that:

- The site is running (See Section 3.1 for help)
- Chrome or Firefox is used to improve performance

3.6.2 Login

Enter the url printed to the launching terminal into your browser to reach the UltraCast e.g. `http://localhost:4000` (port may vary) login page as seen in Figure 2. After filling in your email and password click "SIGN IN" to be redirected to the homepage.

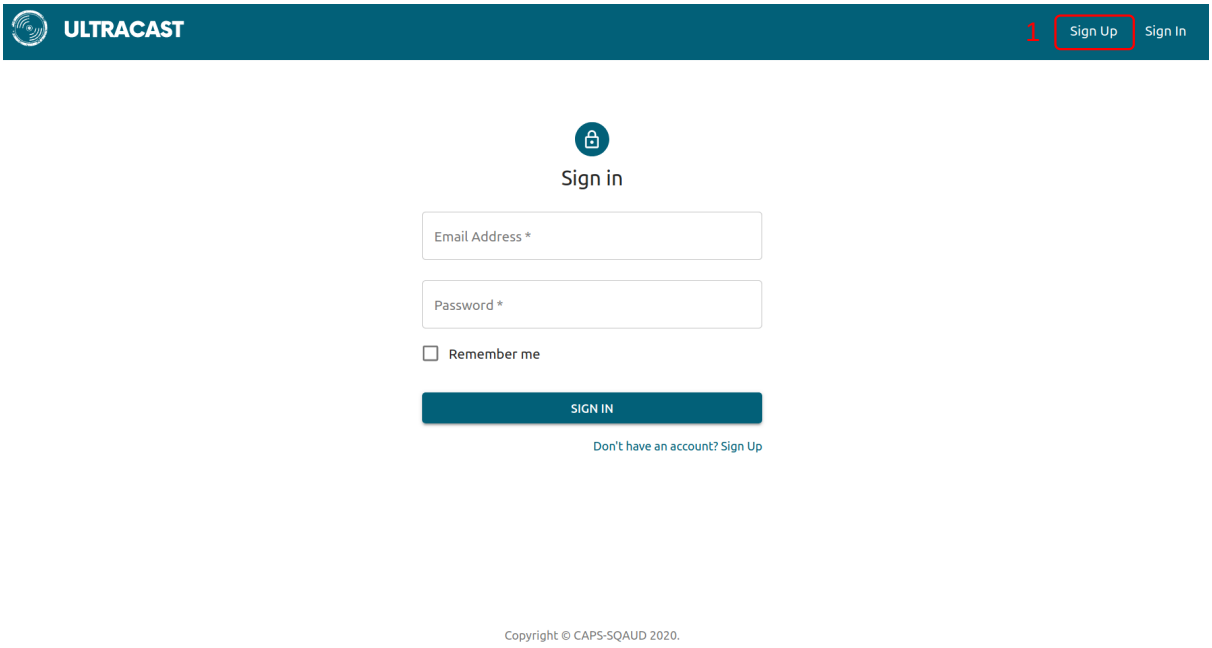

Figure 2: UltraCast Login

If you have not already signed up click on "SIGN UP", labelled with a red "1" in Figure 2, where you will be prompted to fill in your name, email and password before clicking "Sign Up" to login:

3.6.3 Homepage

After logging in you will be directed to the homepage as seen in Figure 3 below. The homepage consists of two components which will be empty if you are a new user:

- **Recommended Podcasts**: Podcasts which our recommendation engine suggests based on your activity
- **Recently Listened**: Podcast Episodes that you have listened to sorted by most recently listened left to right

There are also two important components of UltraCast that can be seen and are labelled in red in Figure 3.

1. **Pages Sidebar**: Lists the pages you can navigate to. The top one is "Home" which directs us to the Homepage
2. **Episode Player**: Player which can be used to play episodes. Described in Section 3.6.5

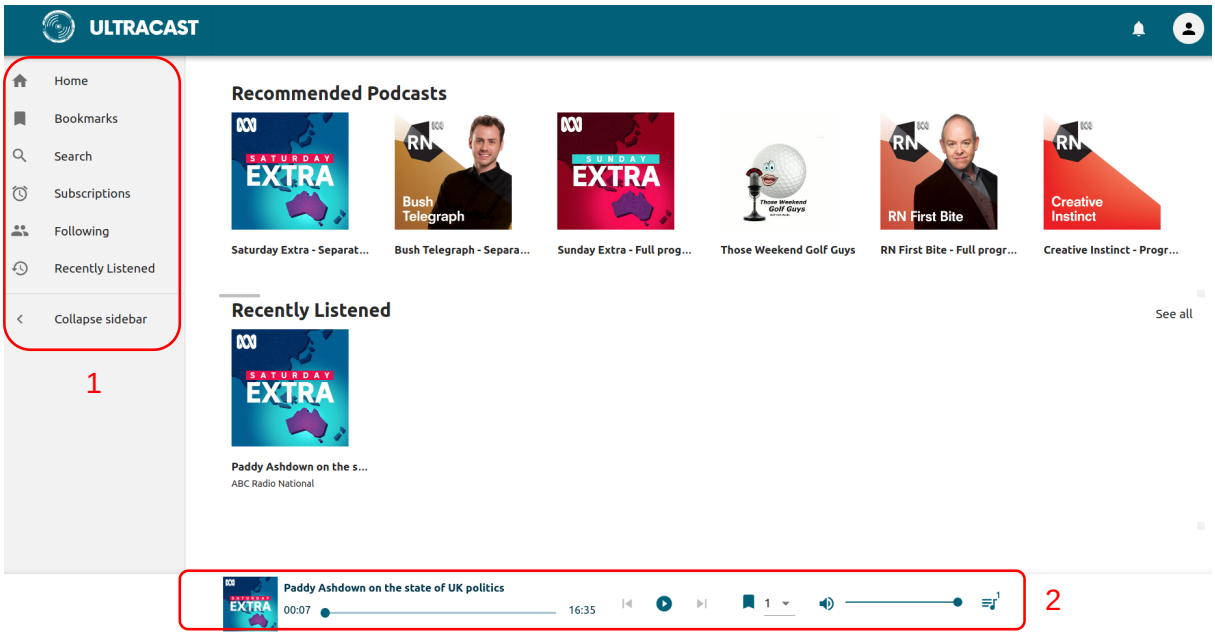

Figure 3: UltraCast Homepage

3.6.4 Searching

To search for a podcast simply click on "Search" in the pages sidebar and type your search term in the search bar which will return matching podcasts as seen in Figure 4. Each "tile" in the search result represents a podcast. The "SAVE SEARCH AS STREAM" button can be used to save a search term. If the search bar is empty then you will be able to view and click on a saved "Stream" to apply that search term.

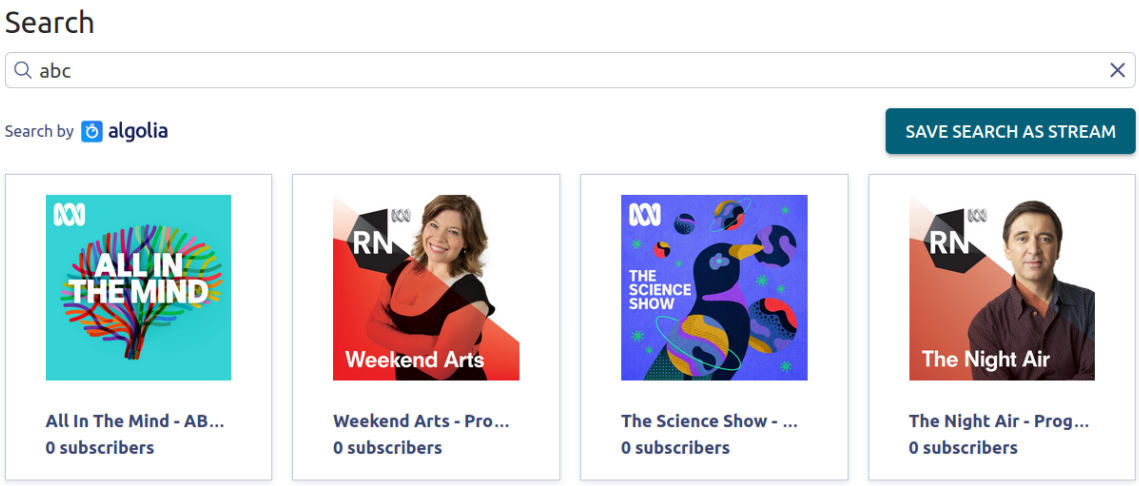

Figure 4: UltraCast Search Result

3.6.5 Playing Podcast Episodes

Once an episode has been clicked on (via the cover image) it will be added to the end of a queue in the "Player", which is labelled "8" in Figure 5. The components of the Episode

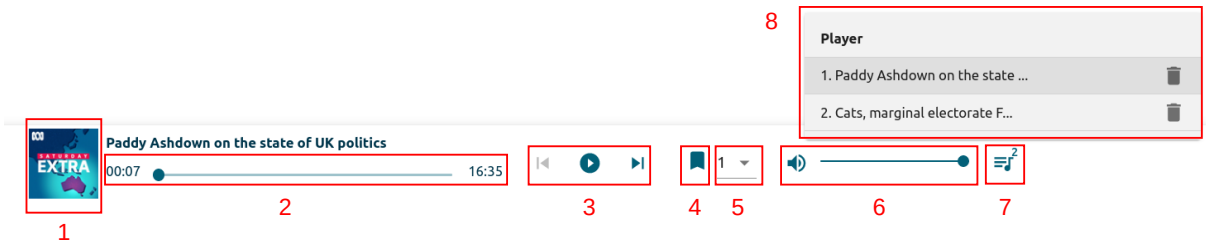

Figure 5: UltraCast Episode Player

Player, marked in Figure 5, are:

1. **Podcast Cover Art** - Click on to view the podcast the episode is part of
2. **Episode Audio Track** - Can "pull and drag" to specific timestamp
3. **Episode Controller** - Start/pause episode and navigate to next/previous in Player
4. **Bookmarks** - Click on to create Bookmark (See Section 3.6.6)
5. **Playback Speed Control**
6. **Volume Control**
7. **Player Toggle** - Toggle the Player pop-up on/off
8. **Player** - Playlist of episodes. Click on episode in player to start it. Click the bin icon to remove an episode. Player will automatically play the next episode once the current one has ended.

3.6.6 Bookmarks

Bookmarks allow you to create a note which is linked to a timestamp in an episode. To create a bookmark , then fill in the title and description.

1. Click the bookmark icon, 4 in Figure 5, when you are at the timestamp in the episode that you to mark
2. Optionally, fill in the title and description in the pop-up
3. Click the "SAVE" button

Bookmarks can be viewed in the Bookmarks page as seen in Figure 6. The numbered labels are described below:

1. Click "Bookmarks" to navigate to the Bookmarks page
2. Click the drop down to show all the bookmarks for a given episode
3. Click to play that episode and jump to the saved timestamp
4. Click to delete the bookmark

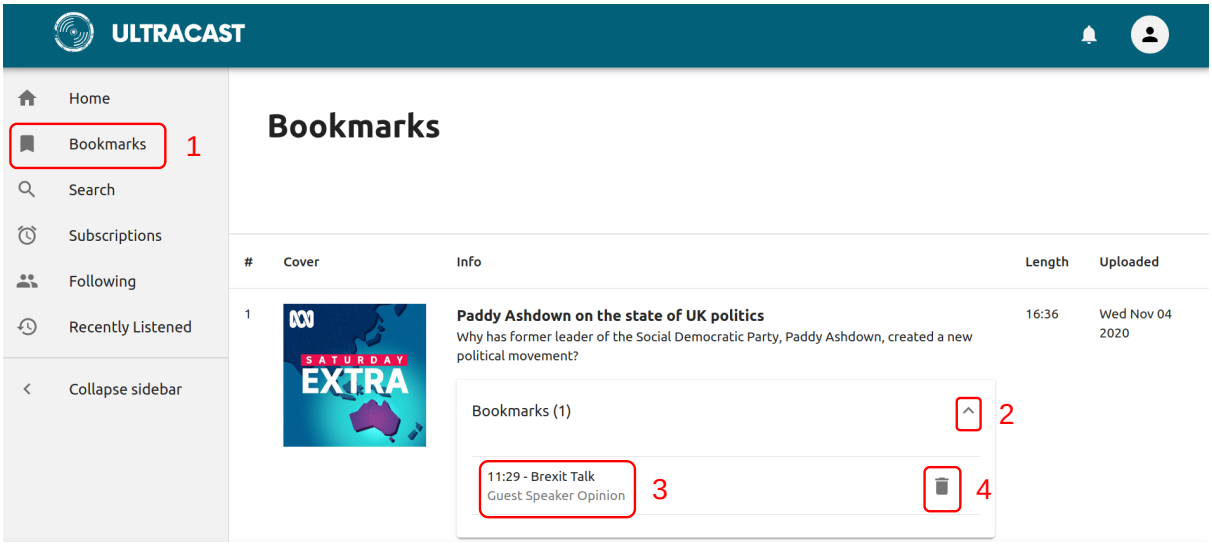

Figure 6: UltraCast Bookmark Page

3.6.7 Subscribing

To follow another user and view their listen history see the steps below with reference to Figure 7:

1. Click "Subscribe" on a podcast
2. Click "Subscriptions" on the pages sidebar to view podcasts your are subscribed to
3. When an episode you are subscribed to uploads a new episode a notification will be displayed on the bell icon

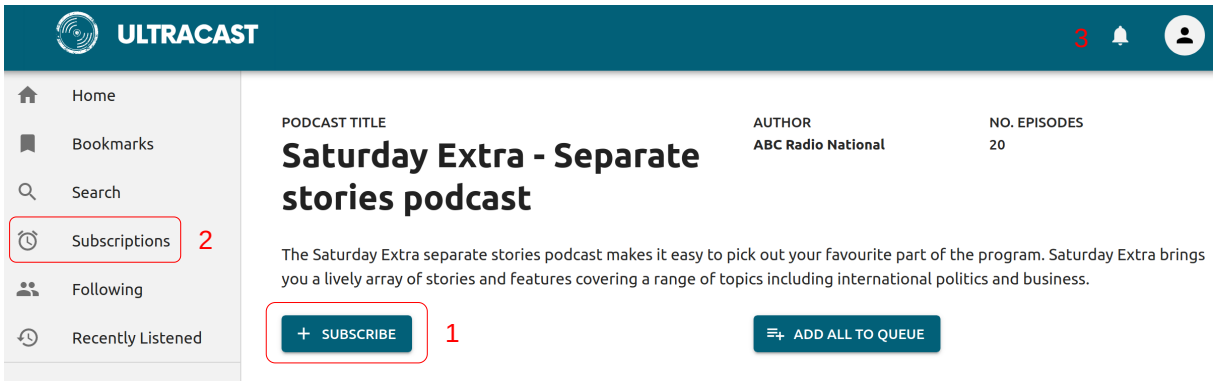

Figure 7: UltraCast Subscribing

3.6.8 Following

To follow another user and view their listen history see the steps below with reference to Figure 8:

1. Click "Following" on the pages sidebar
2. Search user by their email and hit enter
3. Click "Follow" on that user

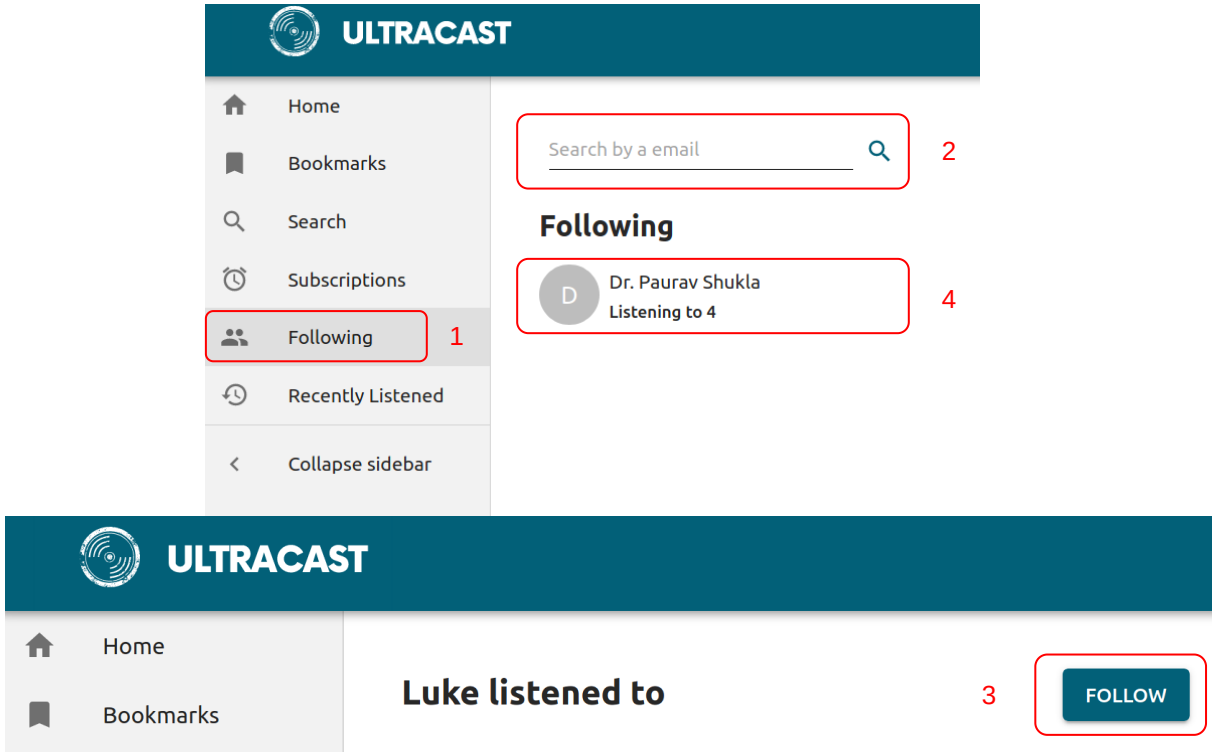

Figure 8: UltraCast Following

3.6.9 Creator / Listener Mode and Logging Out

To logout or swap between creator and listener mode, click on the user icon in the top right corner which will display the drop down shown in Figure 9.

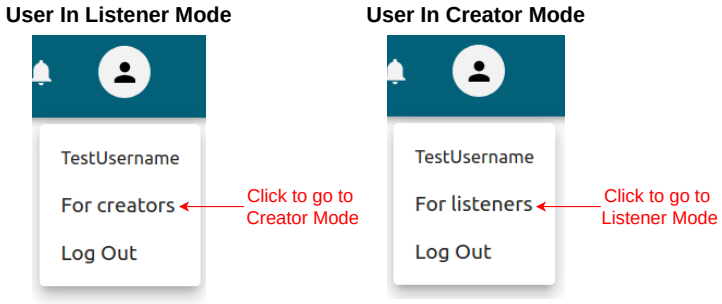

Figure 9: UltraCast User Options

3.6.10 Creator Mode - Creating Podcasts and Episodes

Ensure you are in creator mode as described in Section 3.6.9. As per Figure 10 to create a podcast or episode:

1. Click "Upload" on the pages sidebar
2. In the podcast series bar, select "New Podcast" if you wish to create a new podcast or select an existing podcast if you wish to add a new episode

From this point you will be guided through the creation process.

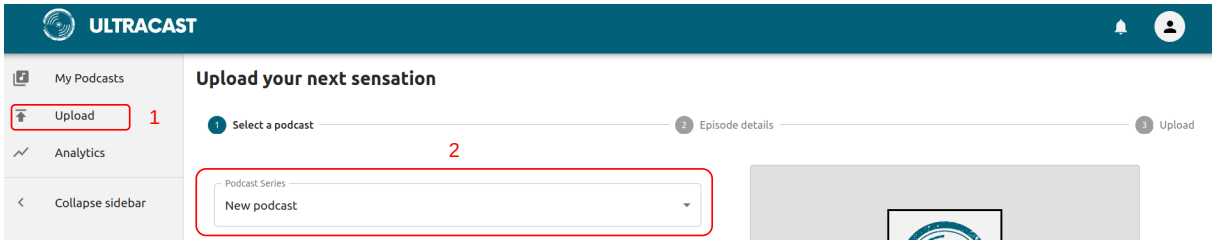

Figure 10: UltraCast Create Podcasts and Episodes

3.6.11 Creator Mode - Editing Podcasts and Episodes

Ensure you are in creator mode as described in Section 3.6.9. To edit a podcast or episode that you have created, as shown in Figure 11:

1. Click "My Podcasts" on the pages sidebar, then click on the podcast you wish to edit
2. Use these buttons to edit the podcast
3. Use these buttons to edit the episode

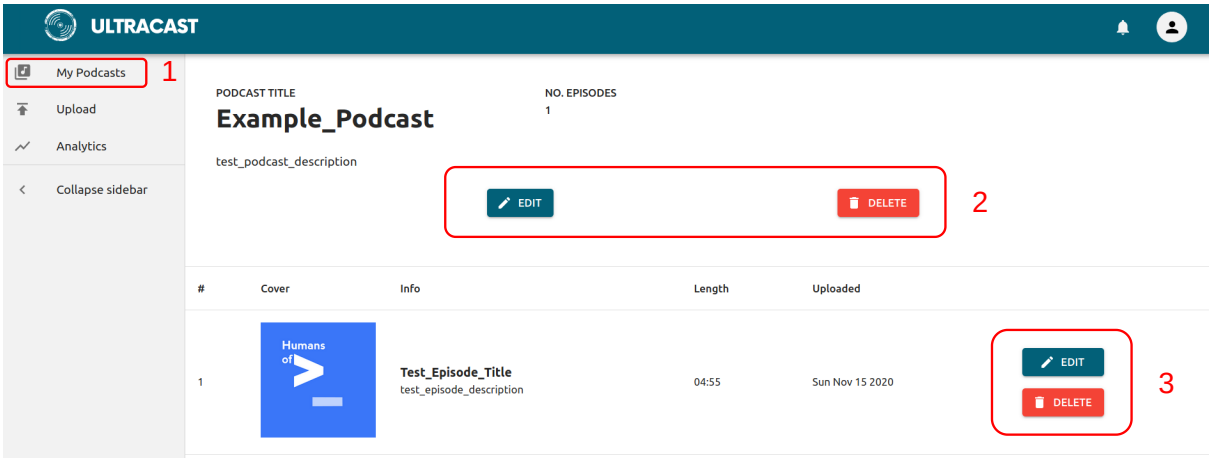

Figure 11: UltraCast Edit Podcasts and Episodes

3.6.12 Creator Mode - Analytics

Ensure you are in creator mode as described in Section 3.6.9. Creator analytics can be accessed as follows, as shown in Figure 12:

1. Click "Analytics" on the pages sidebar
2. Click "Overview" to view podcast/episode metrics
3. Click "Audience" to view a geographical heat-map of viewers

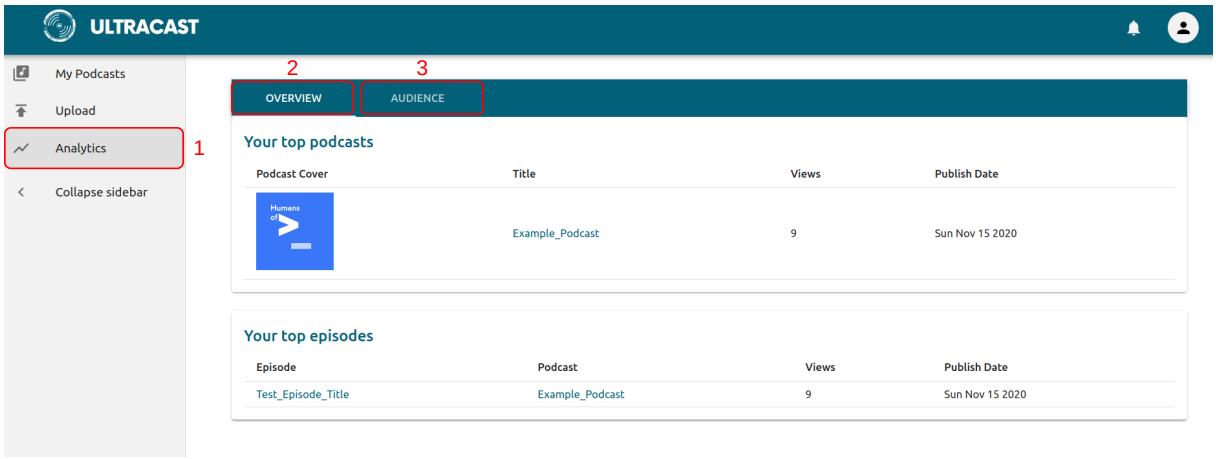

Figure 12: UltraCast Analytics

References

[1] Algolia. https://www.algolia.com/doc/. Accessed: 2020-11-14.

[2] Flask: web development, one drop at a time. https://flask.palletsprojects.com/en/1.1.x/. Accessed: 2020-10-03.

[3] Graphql. https://graphql.org/. Accessed: 2020-10-03.

[4] Algolia. Terms of service. https://www.algolia.com/policies/terms/, April 2018. Accessed: 2020-11-16.

[5] Amazon. Amazon software license. https://aws.amazon.com/asl/, 2020. Accessed: 2020-11-16.

[6] Lambda Foundry Inc. AQR Capital Management LLC and PyData Development Team. License. https://github.com/pandas-dev/pandas/blob/master/LICENSE, 2008. Accessed: 2020-11-16.

[7] Benoit Chesneau and Paul Davis. License. https://github.com/benoitc/gunicorn/blob/master/LICENSE, 2009. Accessed: 2020-11-16.

[8] NumPy Developers. License. https://github.com/pandas-dev/pandas/blob/master/LICENSE, 2005. Accessed: 2020-11-16.

[9] Facebook Inc. React: A javascript library for building user interfaces. https://reactjs.org/. Accessed: 2020-10-03.

[10] Facebook Inc. License. https://github.com/facebook/react/blob/master/LICENSE, Sep 2018. Accessed: 2020-11-16.

[11] Harry Marr. License. https://github.com/MongoEngine/mongoengine/blob/master/LICENSE, 2009. Accessed: 2020-11-16.

[12] MongoDB. The most popular database for modern apps — MongoDB, 2020. http://mongodb.com, Last accessed on 2020-10-03.

[13] Pallets Projects. License. https://flask.palletsprojects.com/en/rtd/license/, 2010. Accessed: 2020-11-16.

[14] Graphene Python. 2015. https://github.com/graphql-python/graphene/blob/master/LICENSE, 2010. Accessed: 2020-11-16.

[15] Gavin Whitner. Podcast statistics (2020). https://musicoomph.com/podcast-statistics/, Sep 2020. Accessed: 2020-10-02.

A Requirements Testing Results

Podcast/Author details & Searching		Pass	Notes
UL-41	As a user, I want to signup to the website, so that I can have a customised experience	Y	empty passwords do not work, empty names work, emails must have an @ in them
UL-14	As a user, I want to be able to login, so that I have a custom experience and can be identified	Y	Can handle invalid email addresses (emails not in db and any other string), remember me button does nothing
UL-24	As a listener, I want to be able to view a title, length, upload date and progress for episodes, so I have a better idea of what to expect before listening	Y	
UL-4	As a listener, I want to be able to view the title, description, author details and list of episodes for a podcast, so I can see if it matches my interests	Y	
UL-3	As a listener, I want to be able to see the total number of subscribers for each podcast returned from searches, so I can see how popular they are	Y	
UL-2	As a listener, I want to be able to use keywords to search for podcasts, so I can find podcasts that interest me	Y	
UL-29	As a listener, I want to be able to save previous searches as "streams", so I can avoid searching for the same thing multiple times	Y	
Podcast Player & Bookmarks			
UL-5	As a listener, I want to be able play episodes, so I can listen to them	Y	
UL-18	As a listener, I want to be able to pause episodes, so I can resume a podcast later when it suits me	Y	
UL-19	As a listener, I want to be able to adjust the volume of episodes, so I can keep volume at a level different to other apps/programs	Y	
UL-20	As a listener, I want to be able to skip to the next, previous episode as well as the start of the current episode, so I can easily navigate between episodes	Y	
UL-21	As a listener, I want to be able to jump to a particular point in an episode, so I can easily navigate within an episode	Y	
UL-22	As a listener, I want to be able to adjust playback speed, so I can listen to episodes at a pace that suits me	Y	
UL-23	As a listener, I want to be able to auto-play episodes within a podcast, so I can easily listen to episodes sequentially	Y	
UL-26	As a listener, I want to be able to "bookmark" points in episodes, with a title and short description, and view bookmarks on a separate page, so I can track of points of interest	Y	
Follow and Recommended			
UL-11	As a listener, I want to be able to sort by most recently played for my episode history, so that I can more easily find episodes I have listened to previously	Y	If you listen to an episode twice, only the most recent view shows up. Hovering over the episode will cause flickering. This is due the page constantly updating in the background.
UL-28	As a listener, I want to be able to follow friends, so I can see what they are listening to	Y	
UL-13	As a listener, I want recommended podcasts to be based on my existing subscriptions, recently played episodes and past searches, so there is a greater chance I will find a podcast that interests me	Y	We should have a specific example in mind to demonstrate how what we've listened to is related to recommended

UL-12	As a listener, I want to be recommended to new podcasts, so I can more easily find new podcasts that interest me	Y	Dashboard top row
Creators			
UL-15	As a content creator, I want to be able to create podcasts, so I can publish episodes under a common group	Y	may take time. recommend not using a file over 5MB and having a file download beforehand
UL-16	As a content creator, I want to be able to delete podcasts, so I can remove podcasts and episodes within	Y	
UL-17	As a content creator, I want to be able to update podcasts, so I can easily add or remove podcast content	Y	Replacing any field with an empty string does not work
UL-27	As a content creator, I want to be able to view episode and podcast metrics; plays, subscribers viewer location, viewer age, so I can better tailor my content	Y	the heatmap uses a local maxima so that the map always contains the full spectrum of colors
Staying up to date			
UL-7	As a listener, I want to be able to subscribe to a podcast, so I can keep up to date with podcasts that interest me	Y	
UL-8	As a listener, I want to be able to unsubscribe to a podcast, so I can stop following podcasts that no longer interest me	Y	
UL-9	As a listener, I want to receive a notification when a new episode for a podcast I am subscribed to is released, so I can keep up to date with episodes that interest me	Y	
UL-30	As a listener, I want to be able to see the latest episode for each subscribed podcast in a "Podcast Subscription Preview" panel, so I can easily keep up to date with the podcasts I am interested in	Y	Subscriptions page on the left sidebar
UL-10	As a listener, I want to be able to view my podcast episode history, so I can review what I have listened to	Y	if an episode is viewed twice, only the most recent view is displayed. If you listen to an episode twice, only the most recent view shows up. Hovering over the episode will cause flickering. This is due the page constantly updating in the background.
UL-6	As a listener, I want episodes I've started playing to be marked as played, so I can keep track of what I have already listened to	Y	items that have not been played are marked with an orange dot on the podcast page only