# Al Gore Rhythms

A playlist app

Eugene Kolo & Braxton Brewton

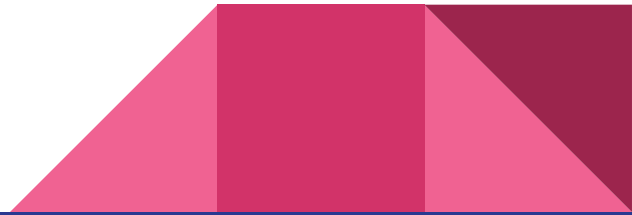
# Summary (1)

- Playlist Application made in Java, JavaScript, and HTML
  - Spark Framework for the web backend
  - Java 8 for the algorithms and data structures
  - jQuery + Bootstrap for the web frontend
  - Gradle + Bash for build/run



# Summary (2)

- Key features:
  - Top 1024 playlists stored
  - Top 8 playlists displayed
  - Autocomplete songs and display top 4 most popular
    - Case insensitive
  - Suggest most popular playlist for a song
  - Add new playlist to the database by song
  - Add up to 128 playlists from a file

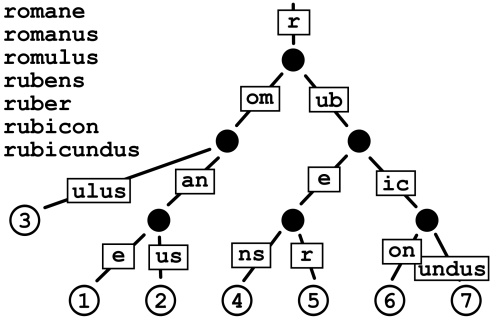


# Data Structure Overview

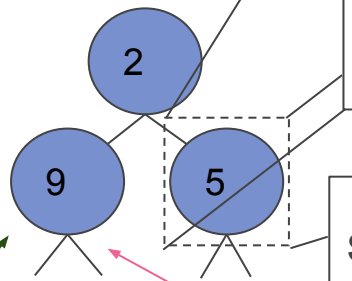
Search song

**AutocompleteDB (PatriciaTrie)**

1 romane
2 romanus
3 romulus
4 rubens
5 ruber
6 rubicon
7 rubicundus




**PlaylistDB (PriorityQueue)**




**Playlist**
- \+ Int: popularity
- \+ List<**Song**>: songList

**Song**
- \+ Int: popularity
- \+ String: title
- \+ String: author
- \+ Playlist: bestPlaylist

Upload playlist

Song ID to song title map (HashMap)

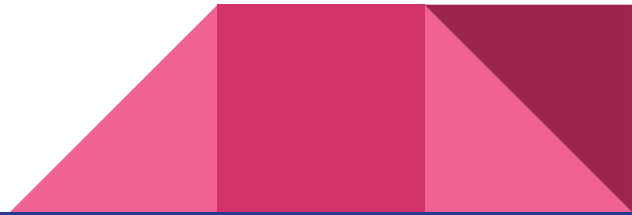
Song title to song object map (HashMap)

Top 8 Playlists Cache (LinkedList)

# Initialization

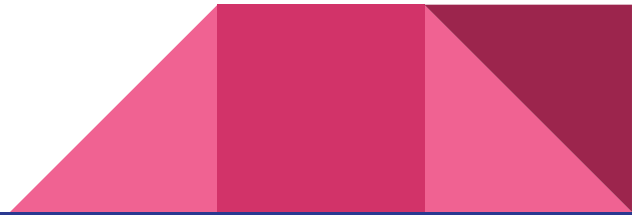
- All songs titles, and authors are marshalled into **Song Objects -** *O(N)*
- Two hash maps, as the **SongDB**, are created: - *O(N)*
  - Song ID to Song Title
  - Song Title to Song Object
- Lower cased, concatenated song title + song author are stored into the **AutocompleteDB** - *O(MN)*
- An empty **PlaylistDB** is initialized - *O(1)*
- Spark web framework starts on localhost

N - Number of Songs, M - Length of longest song



# PlaylistDB

- PlaylistDB is a Min Binary Heap + Linked List
- Utilizes Java's PriorityQueue + LinkedList classes
- Adding playlists is the bulk of the work in the app
  - Minimally popular playlist has to be compared and potentially removed
  - Top 8 cache is potentially updated
  - Songs' popularities are potentially updated
- Time Complexity: (N - # of playlists in PlaylistDB)
  - Inserting a playlist - O(logN)
  - Replacing minimum playlist - O(logN)
  - Updating top 8 - O(1)
  - Returning top 8 - O(1)
  - Finding min - O(1)
  - Updating song's popularity - O(1)
  - Updating song's best playlist - O(1)



# AutocompleteDB

- AutocompleteDB is a Patricia Trie
  - A trie is essentially a tree where each node is a letter and traversing down the tree to leaves will form words
  - A patricia trie is a space optimized trie, where nodes contain pointers to positions in a map where multiple letters per position may be stored.
  - Utilizes Google's PatriciaTrie class
- Time Complexity: (M - character length of song title + author, K - number of songs in DB)
  - Searching for a prefix: O(M)
  - Returning top 4: O(1) amortized. O(KlogK) worst case
  - Suggesting playlist: O(1)

# Web Frontend + Backend

- Requests and responses come as **JSON**
- Uses JavaScript to do HTTP POST and GET **requests** to Spark backend **routes**
- Spark backend responds with a **response**
- Made pretty and **responsive** with Bootstrap

```
POST /api/getAutocomplete
*   Gets a string and searches the database for the most
popular matches
*
*   @note: Case insensitive.
*
*   @req: JSON of "song" matched to partial completion
*       {"song": "obses"}
*   @res: JSON with top 4 most popular autocompleted songs
*       {"0": {
*               "title": "Obsesion",
*               "author": "Roberto Pulido"
*              },
*         "1":
*              {
*               "title":"Obsession Confession",
*               "author": "Slash"
*              }
*           ....
*        }
*
```

```
GET /api/getTop8
POST /api/addPlaylist
POST /api/addPlaylists
POST /api/getAutocomplete
POST /api/suggestPlaylist
```

# Challenges

- Implemented PlaylistDB with a MinMaxBinaryHeap as well
  - Benchmarked and found Min Heap + Cache to be more efficient
  - The MinMaxBinaryHeap's constants were too big
- General programming
  - Had to figure out the difference to a song's popularity a replaced playlist would have
  - Reading Spark documentation
- Frontend user experience
  - Using only the mouse isn't always preferred - added keyboard functionality
  - Typing ID #s to add songs didn't feel natural- added a playlist creation feature
  - Typing capitals in song titles is too much work - made the AutocompleteDB case insensitive
  - Centering things and making them look pretty is hard

# Demo

Project located here: https://github.com/eugenekolo/EC504

# Questions?

- Fun facts:
  - 156 commits
  - 9 coffees
  - The trie was invented by a BU professor
  - Don’t pollute the environment
  - Bubble sort is Eugene’s favourite sort
    - And the hashmap is his favourite structure
  - Merge sort is Braxton’s favorite sort
    - And the arraylist is his favorite structure