Design Project I 2019

poempath.com

Tyler Steele

Introduction

With the advent of the Internet, poetry is more accessible now than ever. With its extensive accessibility, however, comes paradox of choice. Every last person has a form or style of poetry that they would enjoy, but it would take a concentrated effort for them to sort through the variations, authors, topics, et cetera that exist.

What It Is

This web application is a poetry aggregate that suggests poems to users based on their preferences. Users are able to like or dislike each poem, altering the site's future recommendations. The recommendation will be generated using a neural network.

Necessary Features

- User Profiles: Sign Up, Log In, Settings, Preferences
- Poetry Database: Robust poem JSON with author, title, content, tags, style, form...
- Standardized NN Input: Method to rate poems based on a variety of factors that will be compressed down into readable data for the NN
- Functioning NN: The neural network at the heart of this should have a success rate (determined by whether the user likes the recommended poem) of 70% or higher at the time of project completion

Desired/Possible Features

- Curated poetry collections
- User poem submission
- Dictionary/Thesaurus hinting for difficult/rare words or on request

- Typing interface so users may type through poems: Must detect keystrokes and apply them appropriately *User's hands must not leave keyboard post log-in
- Comprehensive typing statistics and lessons
- Random button to ignore user preferences
- Share this poem: enter an email to send a link to the current poem. The page should show just that poem and offer the recipient a chance to register to see more poems. Social media share links could also be provided (preferably not)
- Monetization: This should only be implemented if the user-base is significant enough to generate revenue that could be used to practically benefit the site. This app's structure seems to be suited for a subscription model or donations. No non-literary advertisements (essentially, this will not be a cash cow in any universe so it should not be treated like one)
- Publication Collaboration: Magazines, online publications, publishing houses, et cetera could provide poems to support the site's content in exchange for links to their sites/book sellers. "This poem was provided by XXXXXX, read more from them at XXXXXX." This should not interrupt the flow of the site

Frontend

Vue is lightweight and relatively new to me (and the industry), so it is the ideal candidate for this project's frontend and my learning potential.

Backend

I have never used Koa, but it was recommended to me as being a simple backend framework that works great for small scale applications. This app will not have a complex backend as the majority of heavy listing will take place within the NN.

Neural Network

I plan on building my NN using Keras and Tensorflow (Keras is now integrated into Tensorflow). Apparently, Keras has a simpler user interface that machine learning beginners (me) should benefit from. If I need any of Tensorflow's advanced methods, I can simply use Tensorflow code anywhere within my Keras code (really, it's the other way around). One walkthrough I followed used TensorFlow exclusively and I was able to (mostly) follow the logic. I will use that as a model where applicable: TensorFlow Basics, US Census Data Example

General Design and UX

Upon visiting the site for the first time, user will be asked to provide an email address and password to register. They will then be presented with a list of poems and asked whether they enjoy them. They may also be surveyed on general poetry preferences. This will create an initial profile from which the NN can work. If this ends up being a bad read, the NN should be able to self correct over time, but the longer it takes to understand the user, the worse.

For the first iteration, the page that displays when a user is signed in should always contain nothing but the poem being read and the preference interaction elements (like, dislike). There should be no page-to-page navigation whatsoever- only one view. This means that the decision point resolution of like/dislike should immediately show the next poem without redirection. This likely means that a healthy recommended poem queue should be loaded for the user to provide snappy load time. For example, when viewing a poem, the next five suggestions should be loaded. During that time, the user's newest preference reads will not affect the loaded queue. This is acceptable, but the number of poems in the queue should be calibrated with this in mind.

There should be an expandable user settings option in the primary nav, but the UI should be incredibly simple and streamlined. The goal is to provide poetry, not build a cool web interface. Other options should include features from the *Desired/Possible Features* list if they are implemented. Maybe a "How it works" page about machine learning.

Font choice is imperative. The CSS should be configured such that later on the font can be selected from a list of curated options. For now, a simple, practical font should be used like Roboto. No serifs, cursive, or fantastical fonts. It should almost certainly be monospace to allow for standardized display of the poems.

Post-Completion Addendum

The website is complete, although certain proposal requirements have not been met or have changed. Instead of building a neural network, which would be prohibitively difficult and time-consuming, I utilized an existing, open-source recommendation engine called Raccoon Recommendation engine. It is lightweight and incredibly fast, so there is no need for a preloaded queue or user survey. I did not fulfill any of the “Desired Features,” but those are out of scope for this project and additions I plan on implementing in the coming months. This project has been my most difficult undertaking, and I feel I am a more competent developer for it. Thank you for the opportunity, and for aiding me in this personal accomplishment.

Example Source Code

Vue Code Example – **PoemDisplay.vue**

```
<template>
  <div class="poemWrapper">

    <div class="titleText">{{ title }}</div>
    <div class="authorText">{{ author }}</div>
    <BodyDisplay v-bind:lines="body"></BodyDisplay>
  </div>
</template>

<script>
  import BodyDisplay from './BodyDisplay.vue'

  export default {
    name: 'PoemDisplay',
    props: {
      title: String,
      body: Array,
      author: String,
    },
    components: {
      BodyDisplay
    }
  }
</script>

<style lang="stylus" scoped>
  .poemWrapper
    margin-left 10%
    margin-right 10%
    margin-top 25vmin
    min-height 60vh
    @media only screen and (orientation landscape)
      margin-left 20%
      margin-right 20%
  .titleText
    font-weight bold
    margin-bottom .5rem
    font-size 4rem
    @media only screen and (orientation portrait)
      font-size 2rem
  .authorText
    margin-bottom 1em
    font-size 2.75rem
    @media only screen and (orientation portrait)
      font-size 1.5rem

</style>
```

<template> Here I render *BodyDisplay*, a custom component I made to display the body of a poem

<script> *title*, *body*, and *author* variables are passed down from the parent component (component that renders *PoemDisplay*)

<style> Scoped styling for this specific component. *poemWrapper*, *titleText*, and *authorText* classes are used in the <template>

Koa Code Example – **Dislike Poem Route**

```
132  // Dislike poem - Expects the id of the user and the id of the poem
133  router.post('/dislike', async (ctx) => {
134    const {username, poemID} = ctx.request.body
135    console.log('Disliking poem ' + poemID + ' for username: ' + username)
136    if (username) {
137      if (poemID) {
138        await raccoon.disliked(username, poemID).then(async () => {
139          await raccoon.recommendFor(username, SUGGESTION_RANGE).then(async (results) => {
140            // If there are not any recommended poems
141            if (results.length === 0) {
142              // Return a random one
143              ctx.body = await app.poems.aggregate([{$sample: {size: 1}}])
144              ctx.status = 200
145            } else {
146              let nextPoemIndex = getRandomInt(0, results.length - 1)
147              ctx.body = await app.poems.findOne({_id: ObjectId(results[nextPoemIndex])})
148              console.log('POEM RECOMMENDED!')
149              ctx.status = 200
150            }
151          }).catch(async (error) => {
152            console.log('Raccoon ' + error)
153            // If there is an error, gracefully degrade to random poem
154            ctx.body = await app.poems.aggregate([{$sample: {size: 1}}])
155            ctx.status = 500
156          })
157        })
158      } else {
159        ctx.body = {message: 'poemIDRequired'}
160        ctx.status = 422
161      }
162    } else {
163      ctx.body = {message: 'usernameRequired'}
164      ctx.status = 422
165    }
166  })
```

Raccoon registers the dislike and then provides a recommendation (if enough data is available)

If the recommendation engine has a recommendation, return it. Otherwise, return a random poem

ctx.body and *ctx.status* are returned. Different HTTP codes signify different results.

Server Logs throughout Normal User Flow

Server receives /signUp POST and returns the created user

User likes a poem

User dislikes a poem

Raccoon has enough information to begin recommending poems, rather than returning random poems

```
  <-- POST /signUp
{ message: 'userAdded' }
  --> POST /signUp 200 84ms 28b
  <-- GET /users?username=test2
{ _id: 5c6e269f9983850f1fd7b789,
  username: 'test2',
  lastVote: 1550722719016 }
  --> GET /users?username=test2 200 4ms 91b
  <-- GET /stats
  <-- GET /randomPoem
  --> GET /randomPoem 200 17ms 2.86kb
  --> GET /stats 200 41ms 113.48kb
  <-- OPTIONS /signUp
  --> OPTIONS /signUp 204 0ms
  <-- POST /signUp
{ message: 'userAdded' }
  --> POST /signUp 200 81ms 28b
  <-- GET /users?username=testUser
{ _id: 5c6e28f09983850f1fd7b78a,
  username: 'testUser',
  lastVote: 1550723312375 }
  --> GET /users?username=testUser 200 2ms 94b
  <-- OPTIONS /like
  --> OPTIONS /like 204 0ms
  <-- POST /like
Liking poem 5c5ba0b1e9db64f8413db9f8 for username: testUser
  --> POST /like 200 21ms 3.45kb
  <-- OPTIONS /dislike
  --> OPTIONS /dislike 204 0ms
  <-- POST /dislike
Disliking poem 5c5ba0b1e9db64f8413dc2a9 for username: testUser
  --> POST /dislike 200 23ms 1.32kb
  <-- OPTIONS /dislike
  --> OPTIONS /dislike 204 0ms
  <-- POST /dislike
Disliking poem 5c5ba0b1e9db64f8413dc8da for username: testUser
  --> POST /dislike 200 21ms 827b
  <-- OPTIONS /dislike
  --> OPTIONS /dislike 204 2ms
  <-- POST /dislike
Disliking poem 5c5ba0b0e9db64f8413dafea for username: testUser
  --> POST /dislike 200 21ms 2.08kb
  <-- OPTIONS /dislike
  --> OPTIONS /dislike 204 0ms
  <-- POST /dislike
Disliking poem 5c5ba0b1e9db64f8413dc74f for username: testUser
POEM RECOMMENDED!
  --> POST /dislike 200 30ms 948b
  <-- OPTIONS /like
  --> OPTIONS /like 204 0ms
  <-- POST /like
Liking poem 5c5ba0b2e9db64f8413dd860 for username: testUser
POEM RECOMMENDED!
  --> POST /like 200 33ms 1.16kb
  <-- OPTIONS /like
  --> OPTIONS /like 204 0ms
  <-- POST /like
Liking poem 5c5ba0b2e9db64f8413dd819 for username: testUser
POEM RECOMMENDED!
  --> POST /like 200 29ms 2.24kb
  <-- OPTIONS /like
  --> OPTIONS /like 204 2ms
  <-- POST /like
```