Mosaic - SWDP

Problem Definition

Known Facts	User Requirements	Necessary Processing	Alternative Solutions
• Application has one search bar, and when a query is entered, the program will fetch and display relevant content from YouTube, Reddit, and TikTok • Tech stack will include the following technologies: HTML, CSS, JavaScript, Node.js, MongoDB, Heroku, EJS, Express, Mongoose, Node-Fetch, Random-Words, TikTok-Scraper, Dotenv, Nodemon, Bootstrap 5, Google Fonts, jQuery • Content will be fetched by making GET requests to endpoints provided by the YouTube Data API, Reddit API, and TikTok API. The returned data can then be deserialized,	• Filter social media platforms by checking/unchecking boxes corresponding to each platform • Customize the max posts per platform by selecting a number from 1–20 in the relevant labelled dropdown menu • Enter a search query in the provided search bar and press the button “Let’s go!” in order to submit your query; alternatively, enter * to generate a random search query or $ to search for trending content	**server.js** If not in a production environment, use local environment variables. Require express, ejs, and mongoose NPM packages. Initialize Express application and configure options. Connect application to MongoDB database. If there is an error in the connection, then output the error to the console. Otherwise, if the connection is successful, then output to the console that the program is connected to the database. Hook up route to index page. Listen on the port provided by Heroku or port 3000, and output to the console that the server has started. **customization.js** Require mongoose NPM package. Construct a schema to outline what/how data will be stored in the database:	**server.js** If not in a production environment, use local environment variables. Require express, ejs, and mongoose NPM packages. Initialize Express application and configure options. Connect application to MongoDB database. If there is an error in the connection, then output the error to the console. Otherwise, if the connection is successful, then output to the console that the program is connected to the database. Hook up route to index page. Listen on the port provided by Heroku or port 3000, and output to the console that the server has started. **customization.js** Require mongoose NPM package. Construct a schema to outline what/how data will be stored in the database:

and certain properties can be injected into HTML embeds, which will be inserted into a div element on the index page • The data returned from the APIs should be output to both the terminal console and the Element Inspector console • The program must pass in a registered API key as a query parameter when making GET requests to the YouTube Data API endpoints • The Reddit API endpoints can be accessed without authorization • The TikTok API will be communicated with indirectly via a third-party scraper • User inputs will be captured in an HTML form which makes a POST request to the index route when submitted; this data can then be accessed in the POST request route in the backend, with the route		• A mandatory selected platforms setting, of type Array, that has a default value of all social media platforms set to true • A mandatory max posts per platform setting, of type Number, that has a default value of 2 • A mandatory creation date setting, of type Date, which has a default value of the time at which the database document was created Export model derived from schema. **index.js** Create router object and require NPM packages express, node-fetch, tiktok-scraper, and random-words; also require Customization model. Set up a GET route to the home page: • Extract search query found in query string of URL • Save YouTube API key • Retrieve settings from database • If no settings could be	• A mandatory selected platforms setting, of type Array, that has a default value of all social media platforms set to true • A mandatory max posts per platform setting, of type Number, that has a default value of 2 • A mandatory creation date setting, of type Date, which has a default value of the time at which the database document was created Export model derived from schema. **index.js** Create router object and require NPM packages express, node-fetch, tiktok-scraper, and random-words; also require Customization model. Set up a GET route to the home page: • Extract search query found in query string of URL • Save YouTube API key • Retrieve settings from database • If no settings could be

transferring the search query to the GET route via a query string and saving the rest of the data to the database • Application will be deployed to Heroku and connected to a cloud database offered by MongoDB Atlas • Application will also be available on localhost:3000 for development and testing purposes • Application will be fully mobile responsive • User will be able to filter the social media platforms by checking/unchecking boxes which appear above the search bar • User will be able to randomize their content feed by typing * in the search bar • User will be able to configure the max posts per platform by selecting a number from 1–20 in a labelled dropdown menu • Content will populate on the		retrieved, then create a document holding the default settings and save it to the database • Retrieve the user's max posts per platform option • If there is a search query: ○ If the search query is a *, then set the search query to a random word ○ If the user selected to see YouTube content, then fetch YouTube data, either fetching trending videos if the search query is a $ or fetching videos matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see Reddit content, then fetch Reddit data, either fetching trending	retrieved, then create a document holding the default settings and save it to the database • Retrieve the user's max posts per platform option • If there is a search query: ○ If the search query is a *, then set the search query to a random word ○ If the user selected to see YouTube content, then fetch YouTube data, either fetching trending videos if the search query is a $ or fetching videos matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see Reddit content, then fetch Reddit data, either fetching trending

page in 5-second intervals ● The user can query for trending content by entering $ in the search bar		posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see TikTok content, then fetch TikTok data, either fetching trending posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Since the data comes parsed, simply stringify it and output the parsed version to the console ● Render the index page, passing in the search query, stringified data, and settings from the database Set up a POST route to	posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see TikTok content, then fetch TikTok data, either fetching trending posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Since the data comes parsed, simply stringify it and output the parsed version to the console ● Render the index page, passing in the search query, stringified data, and settings from the database Set up a POST route to

		the home page: • Extract user search query from the form they submitted • Retrieve settings from database • Read values of checkbox inputs submitted in the form and, based on those values, update the user's selected platforms setting • Update the user's max posts per platform setting to reflect the value they submitted • Save the updated settings to the database • If there is a valid search query, then redirect to home route with query string appended • Otherwise, redirect to regular home route Export router. **script.js** Retrieve a Node List of hidden inputs from the DOM (Document Object Model), convert this Node List to an array, and return a new array where each element is the value of the hidden input from the old array. This new array stores the stringified data corresponding to each of	the home page: • Extract user search query from the form they submitted • Retrieve settings from database • Read values of checkbox inputs submitted in the form and, based on those values, update the user's selected platforms setting • Update the user's max posts per platform setting to reflect the value they submitted • Save the updated settings to the database • If there is a valid search query, then redirect to home route with query string appended • Otherwise, redirect to regular home route Export router. **script.js** Save the values of hidden inputs from the DOM, where the hidden inputs each store the stringified data from a particular social media platform. If there is non-empty string YouTube data, then parse the data, save the parsed version of the

		the social media platforms. For each element in this new array: • If the element is non-empty string data, then parse the data and output the parsed data to the console • Regardless of whether the element is non-empty string data, push the element to another array called parsedData Assign elements in parsedData array to respective individual variables. Select results section from the DOM. If the user checked off one or more social media platforms: • Select spinner from DOM and make it visible • Retrieve the value of the DOM select element • Load in social media content in 5-second intervals, keeping in mind the user's max posts per platform setting and hiding the loading spinner right before the	data, and output the parsed data to the console. If there is non-empty string Reddit data, then parse the data, save the parsed version of the data, and output the parsed data to the console. If there is non-empty string TikTok data, then parse the data, save the parsed version of the data, and output the parsed data to the console. Select results section from the DOM. If the user checked off one or more social media platforms: • Select spinner from DOM and make it visible • Retrieve the value of the DOM select element • Load in social media content in 5-second intervals, keeping in mind the user's max posts per platform setting and hiding the loading spinner right before the first content appears on the page Otherwise: • Insert a paragraph into

		first content appears on the page Otherwise: ● Insert a paragraph into the results div instructing the user to select one or more social media platforms to see results (End of program is absent, as it will continue running until the server is shut down.)	the results div instructing the user to select one or more social media platforms to see results (End of program is absent, as it will continue running until the server is shut down.)

Analysis (IPO Chart)

Input Data	Processing Steps	Output Data
● Checked/not checked values for the social media platform filtering checkboxes. Checking a box results in an input value of 'on' while unchecking a box results in an input value of undefined. ● A number from 1–20 denoting the max posts per platform ● A search query typed in by the user	**server.js** If not in a production environment, use local environment variables. Require express, ejs, and mongoose NPM packages. Initialize Express application and configure options. Connect application to MongoDB database. If there is an error in the connection, then output the error to the console. Otherwise, if the connection is successful, then output to the console that the program is connected to the database. Hook up route to index page.	● Navigation bar with Mosaic logo ● Hero section with illustration on the right side and text on the left side ● A call-to-action for the user to enter a search query ● Checkboxes to filter the social media platforms ● A labelled dropdown to customize the max posts per platform ● A search bar, with placeholder text, and corresponding button to enter a search query ● Faint text beneath the

	Listen on the port provided by Heroku or port 3000, and output to the console that the server has started. **customization.js** Require mongoose NPM package. Construct a schema to outline what/how data will be stored in the database: ● A mandatory selected platforms setting, of type Array, that has a default value of all social media platforms set to true ● A mandatory max posts per platform setting, of type Number, that has a default value of 2 ● A mandatory creation date setting, of type Date, which has a default value of the time at which the database document was created Export model derived from schema. **index.js** Create router object and require NPM packages express, node-fetch, tiktok-scraper, and random-words; also require Customization model. Set up a GET route to the home page: ● Extract search query found in query string of URL ● Save YouTube API key ● Retrieve settings from database ● If no settings could be retrieved, then create a document holding the default settings and save it to the database ● Retrieve the user's max	search bar informing the user of extra commands ● A results section which is empty by default ● A grey sticky footer, with light grey text indicating copyright information ● A warning message if the user tries to submit the form without entering anything into the search bar ● Animated loading spinner while data is being fetched ● Bold text indicating the search query after the user has submitted the form ● YouTube, Reddit, and TikTok embeds loaded into the page in 5-second intervals ● API data in both the terminal console and Element Inspector console

	posts per platform option ● If there is a search query: ○ If the search query is a *, then set the search query to a random word ○ If the user selected to see YouTube content, then fetch YouTube data, either fetching trending videos if the search query is a $ or fetching videos matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see Reddit content, then fetch Reddit data, either fetching trending posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Parse this data, stringify it, and output the parsed version to the console ○ If the user selected to see TikTok content, then fetch TikTok data, either fetching trending posts if the search query is a $ or fetching posts matching the query the user typed in otherwise. Since the data comes parsed, simply stringify it and output the parsed version to the console ● Render the index page, passing in the search query, stringified data, and settings from the database	

	Set up a POST route to the home page: ● Extract user search query from the form they submitted ● Retrieve settings from database ● Read values of checkbox inputs submitted in the form and, based on those values, update the user's selected platforms setting ● Update the user's max posts per platform setting to reflect the value they submitted ● Save the updated settings to the database ● If there is a valid search query, then redirect to home route with query string appended ● Otherwise, redirect to regular home route Export router. **script.js** Retrieve a Node List of hidden inputs from the DOM (Document Object Model), convert this Node List to an array, and return a new array where each element is the value of the hidden input from the old array. This new array stores the stringified data corresponding to each of the social media platforms. For each element in this new array: ● If the element is non-empty string data, then parse the data and output the parsed data to the console ● Regardless of whether the element is non-empty string data, push the element to another array	

	called parsedData Assign elements in parsedData array to respective individual variables. Select results section from the DOM. If the user checked off one or more social media platforms: ● Select spinner from DOM and make it visible ● Retrieve the value of the DOM select element ● Load in social media content in 5-second intervals, keeping in mind the user's max posts per platform setting and hiding the loading spinner right before the first content appears on the page Otherwise: ● Insert a paragraph into the results div instructing the user to select one or more social media platforms to see results (End of program is absent, as it will continue running until the server is shut down.)	

Design Using Pseudocode

server.js

```
If (node environment is NOT equal to production):
        Require('dotenv').config()

Initialize constant variable express to require('express')
Initialize constant variable ejs to require('ejs')
Initialize constant variable mongoose to require('mongoose')

Initialize constant variable app to express()
Call set function on app variable, passing in arguments ('view engine', 'ejs')

Call use function on app variable, passing in argument (express.urlencoded({extended: true}))

Call use function on app variable, passing in argument (express.static('public'))

Call mongoose.connect function, with arguments (DATABASE_URL env variable, {
        useNewUrlParser set to true
        useUnifiedTopology set to true
})

Initialize constant variable db to mongoose.connection
Call db.on function, passing in arguments ('error', (err) => output error to console)
Call db.once function, passing in arguments ('open', () => output successful database connection message to console)

Initialize constant variable indexRouter to require('./routes/index')
Call use function on app variable, passing in arguments ('/', indexRouter)

Call listen function on app variable, passing in arguments (PORT environment variable OR 3000, () => output to console that server has started)
```

index.js

```
Initialize constant variable express to require('express')
Initialize constant variable router to express.Router()
Initialize constant variable Customization to require('../models/customization')
Initialize constant variable fetch to require('node-fetch')
Initialize constant variable TikTokScraper to require('tiktok-scraper')
Initialize constant variable randomWords to require('random-words')

GET router with '/' as URL path and the following asynchronous callback function, which has parameters (req, res):
        Try:
                Initialize variable searchQuery to req.query.q
                Initialize constant variable apiKey to API_KEY environment variable

                Declare variables returnedYoutubeData, parsedYoutubeData, and stringYoutubeData
                Declare variables returnedRedditData, parsedRedditData, and stringRedditData
                Declare variables parsedTiktokData, stringTiktokData

                Initialize variable settings to awaited document from customizations collection

                If (no document found):
                        Set settings to a new document based on Customization model
                        Await settings.save()
```

```
            Initialize constant variable maxResults to settings.maxPostsPerPlatform

            If (searchQuery):
                If (searchQuery is equal to "*")
                    Set searchQuery to a random word by calling randomWords function

                If (item at index 0 of settings.selectedPlatforms array is equal to true):
                    If (searchQuery is equal to "$"):
                        Set returnedYoutubeData to awaited return value of calling the YouTube Data API for most popular videos
                    Else:
                        Set returnedYoutubeData to awaited return value of calling the YouTube Data API for videos matching query

                    Set parsedYoutubeData to awaited parsed version of returnedYoutubeData
                    Set stringYoutubeData to stringified version of parsedYoutubeData
                    Output parsedYoutubeData to console

                If (item at index 1 of settings.selectedPlatforms array is equal to true):
                    If (searchQuery is equal to "$"):
                        Set returnedRedditData to awaited return value of calling the Reddit API for most popular posts
                    Else:
                        Set returnedRedditData to awaited return value of calling the Reddit API for posts matching query

                    Set parsedRedditData to awaited parsed version of returnedRedditData
                    Set stringRedditData to stringified version of parsedRedditData
                    Output parsedRedditData to console

                If (item at index 2 of settings.selectedPlatforms array is equal to true):
                    If (searchQuery is equal to "$"):
                        Set searchQuery to 'trending'

                    Set parsedTiktokData to await TikTokScraper.user(searchQuery, {
                        Set number to maxResults
                        Set sessionList to ['sid_tt=58ba9e34431774703d3c34e60d584475;']
                    })

                    Set stringTiktokData to stringified version of parsedTiktokData
                    Output parsedTiktokData to console

            Res.render index page, with context object including: searchQuery, stringYoutubeData, stringRedditData, stringTiktokData, settings

        Catch (err):
            Output err to console

POST router with '/' as URL path and the following asynchronous callback function, which has parameters (req, res):
    Initialize constant variable searchQuery to req.body.searchQuery
    Initialize constant variable settings to awaited document from customizations collection

    Initialize constant variable youtubeCheckbox to true if req.body.youtubeCheckbox equals 'on' and false otherwise
    Initialize constant variable redditCheckbox to true if req.body.redditCheckbox equals 'on' and false otherwise
    Initialize constant variable tiktokCheckbox to true if req.body.tiktokCheckbox equals 'on' and false otherwise

    Set settings.selectedPlatforms to [youtubeCheckbox, redditCheckbox, tiktokCheckbox]
    Set settings.maxPostsPerPlatform to req.body.maxPostsPerPlatform

    Await settings.save()

    If (searchQuery):
        Res.redirect to '/' route with query string appended that denotes the search query
```

Else:
Res.redirect to '/' route

Set module.exports to router

Testing/Verification

Test Case 1: Mobile responsiveness

This test case involved using the Element Inspector to resize the application window. As expected, after a certain breakpoint, the hero illustration fell beneath the bold white text, ensuring that the elements would appear organized on a mobile display. All other sections continued to appear readable and structured when the window size was decreased, ensuring that the Mosaic application is fully mobile responsive.

Test Case 2: Form validation

This test case involved leaving the search bar blank and attempting to submit the form. As expected, a warning message was displayed, informing the user to fill out the field in order to proceed. This proves that Mosaic's form validation is working as intended.

Test Case 3: Data persistence

This test case involved changing the social media filtering options as well as the max posts per platform and submitting sample search queries. As expected, when the page was refreshed, the options remained changed, indicating that the user's settings are being persisted. Additionally, the Mosaic page was exited and reopened; as expected, the re-opened page persisted the user's custom settings.

Test Case 4: No social media platforms selected

This test case involved unchecking all of the social media platforms and submitting sample search queries. As expected, the following message was displayed in the results section: "Select one or more social media

platforms to see results." This confirms the intended functionality, as no posts should be returned if there are no social media platforms to pull from.

Test Case 5: Indicating search queries

This test case involved submitting sample search queries and paying attention to both the URL and the header of the results section. As expected, when valid search queries were submitted, such as "dog" or "cat", the query string in the URL reflected this, and the results section displayed the following header: "Results for x", where x represents the search query in quotation marks. Deliberate exceptions to this rule include when the special commands are entered as search queries—namely, * and $. When * was entered, the query string reflected the * symbol while the results header showed the random keyword generated; when $ was entered, the query string reflected the $ symbol while the results header read "Trending results." All of this behaviour is to be expected, so Mosaic is indicating search queries appropriately.

Test Case 6: Settings–content consistency

This test case involved trying out various combinations of settings and submitting sample search queries to see if the displayed results would be tailored accordingly. For example, would ticking only YouTube result in only YouTube content being displayed? What about ticking only Reddit and TikTok? If I changed the max results per platform to 1, would I only get 1 result per selected platform? As expected, the results were always filtered in alignment with the settings.

Test Case 7: Special commands

This test case involved entering the special commands as search queries. As expected, when * was entered, a randomly-generated word became the new search query; when $ was entered, the results header showed "Trending results" and trending results were indeed displayed.

Test Case 8: Embed interactivity

This test case involved clicking on the social media posts after they were loaded in to ensure that a user could play the content. As expected, the embeds responded to user action, and videos would play when clicked.

Test Case 9: Console data

This test case involved submitting sample search queries and observing the activity in the console. As expected, when search queries were submitted, objects containing the parsed API data were logged to the console, confirming the intended functionality.

Maintenance

While it is too early to practice maintenance, as Mosaic has only just been deployed, strategies for maintenance can be discussed. One strategy for maintenance could be performing weekly tests and code reviews as opposed to simply deploying the application and forgetting all about it! This way, bugs in the code can not only be addressed but addressed quickly, ensuring a well-maintained program. Furthermore, all updates and modifications to the codebase can be committed to the remote GitHub repository. This way, there is a record of changes that can be referred to in times of confusion; equally important, old commits can be reverted to in case any updates backfire.

Maintenance would be pretty boring if it only meant fixing bugs and keeping the status quo—it may also entail new and exciting features! Some features that may be implemented in future iterations of Mosaic include:

- More social media platforms, such as Instagram and Twitter
- AJAX for a more dynamic user experience
- Incorporating individual user accounts
- Improving querying speed

Finally, when might it be time to retire the program? Arguably, there was no need for the program in the first place, so retirement due to lack of need might not be applicable. However, if the social media APIs permanently changed to prohibit certain content from being fetched, then that may render a program such as Mosaic infeasible. Furthermore, many free APIs have definite call limits, and so if Mosaic exhausts the call limits, then it may no longer be able to work. Finally, there is a legal concern of whether it is right for a developer to profit off of an application that steals content from other platforms. If the laws change, then so may the longevity of Mosaic.