Name:  **Kingsley E. Abia**

Technologies used: VueJS (FrontEnd), Laravel (BackEnd) & MySQL (Database).

**Task 1 Solution:**

**Brief Solution Overview:**

On frontend page load, 2 API calls are made to the backend:

* First API call is to import the two news feeds from the resources section (Newsfeed 1 and Newsfeed 2) into the database.
* Second API call is to fetch the newly imported feeds from the database to display to the client on the frontend.

By default, the tech news feed is loaded on page load, but you can switch to the Europe feed by clicking the Europe button. You can search the feed via article title.

**Technical Details:**

* **Frontend:** The techfeed.vue and europefeed.vue files display both tech and Europe newsfeeds respectively. To implement abstraction since both techfeed.vue and europefeed.vue files have a lot of code in common, a feedskeleton.vue file was created. Both files import the feedskeleton.vue component and supply feed title and feed content as arguments.
* The computed block handles the article search by filtering the feed and only displaying to the user the articles with the search words/phrases the user enters in the search box.
* A 404-error page is displayed if a route which does not exist. This logic is handles in the main.js file.
* **Backend:** The newsfeedController.php file handles both the importation of new content to the database and the fetching of the imported content for the frontend display.
  + The importNewsFeeds() function handles the importation of new content by first converting the XML response got from the resource API to JSON format so it can be easily stored in the database.
  + To implement abstraction since both tech and europe channel structure are identical, manage\_channels.php and manage\_feed.php files located in the Helpers folder were created. They manage the importation and insertion of new feed content into the database.
  + I concatenated the UNIX timestamp as at the point of creation and the for-loop number count ($i) to create the feed\_id for each article. This way, they are unique, and no two articles would share the same ID.
  + Model instances were saved in a DB transaction so if any error occurs, every commit gets rolled back. This ensures uniformity.
  + Before any article is inserted into the DB, the guid of the article is searched to determine if it already exists. If it does, then the article is skipped, if it doesn’t, then the article is inserted into the DB thus no duplication of articles in the DB.
  + The getNewsFeed($feedType) function takes a variable feedType as an argument and returns the corresponding feed for the feedType argument supplied.

**Assumptions:**

* The guid of a newsfeed item is unique and no 2 feed items share the same guid.
* The channel description array would either be empty or not contain more than one item.

**Task 2 Solution:**

**Brief Solution Overview:**

The user clicks on the star on any article to be rated and boom! That rating is registered in the database. The user can only rate an article only once unless the localstorage is cleared or the program is run on another browser as the articles rated are being tracked in the localstorage of the browser.

If a user tries to rate an article more than once, an alert pops up alerting the user that the most recent rating wouldn’t be registered.

**Technical Details:**

* **Frontend:** Once a user clicks a star to rate an article, the updateRating() method in the feedskeleton.vue is supplied with the rating and the feed\_id of the article. The method then checks if the article has been previously rated by the user by checking the localstorage for the feed\_id of the article. If it does not find it, it sends the rating and feed\_id to the backend to store in the database before then storing the feed\_id in the localstorage to prevent the user from rating that same article again. If it does find the feed\_id in the localstorage, an alert saying the user has rated the article is displayed.
* **Backend:** The ratingsController.php file handles all the rating backend logic. The method saveRatings() receives a POST HTTP request which has a payload containing both the rating and the feed\_id. It checks if the article has been rated before, if yes, it adds the new rating to the previous rating. If no, it creates a new row in the ratings table.

**Assumptions:**

* The user would not clear the localstorage of the browser and will not open the feed on another browser after rating an article. This is to prevent multiple ratings for an article by the same user.

**Task 3 Solution:**

**Brief Solution Overview:**

The endpoint to be used to access the 5 best-rated articles is [http://0.0.0.0:9000/api/ratings/topfive](http://localhost:8000/api/ratings/topfive)

The results for the endpoint are static for 5 minutes after which fresh data is loaded.

**Technical Details:**

* **Backend:** The getBestRatedArticles() function handles this by returning a response which is updated every 5 minutes if constantly hit with requests.

So, I figured, instead of running a cronjob every 5 minutes to update the last\_requests table, I could utilize its created\_at field which also gets populated when a new article breaks into the top 5 best-rated articles. So, each time a user hits the endpoint with a request, it checks if the difference between the last request and the current datetime is greater than 5 minutes. If yes, it generates a new list and stores it in the last\_requests table, if no, it just retrieves the previous list, orders it by descending order so it picks the last inserted list and sends it back as a response.

This way, system and database resources are conserved as running a cronjob every 5 minutes to update a table is very resource-intensive.

**Task 4 Solution:**

**Brief Solution Overview:**

The user loads up the techadmin page (<http://0.0.0.0:8080/techadmin>) and it first loads up tech articles from the database and then every 30 minutes, it checks for new articles from the Tech Resource Feed API. If a new article exists, an alert pops up giving the user an option to load and display fresh articles on the feed. Same concept applies to the europeadmin page (<http://0.0.0.0:8080/europeadmin>).

**Technical Details:**

* **Frontend:** Just like the tech and Europe feeds, the techadmin.vue and europeadmin.vue files displays the techadmin and europeadmin pages respectively. To implement abstraction since both techadmin.vue and europeadmin.vue files have a lot of code in common, an adminfeedskeleton.vue file was created. Both files import the adminfeedskeleton.vue component and supply feed title and feed content as arguments.

On techadmin page load, the Vue instance is mounted and then the getNewsFeed('Tech') function is executed to get the current articles from the database then after every 30 minutes, it checks for new articles from the Tech Resource Feed API. If there is, i.e. if the current feed count is lesser than the feed count from the resource API, the component newfeedalert becomes visible. Abstraction is also utilized here by outsourcing the implementation details of the alert to the newfeedalert.vue file. Clicking the load new content button triggers a reload of the page which now displays the new articles on top.

* **Backend:** Once new articles are available and the importNewsFeed() function is called, an API call is made to the importNewsFeeds() function in the newsfeedController.php to import new articles from the resource APIs. It only inserts a new article whose guid isn’t already in the database. On reload of the page, the getNewsFeed($feedType) function is executed and the new content is sent along with old content as a response to the frontend to display appropriately.

**Assumptions:**

* The guid of a newsfeed item is unique and no 2 feed items share the same guid.
* The channel description array would either be empty or not contain more than one item.

**Steps to run my solutions:**

Assuming Docker is installed on your local machine.

* **Frontend:** Navigate into the frontend folder, open your terminal and run the following commands:
* docker build -t frontend .
* docker run -it -p 8080:80 --rm --name dockerize-frontend frontend
* **Backend:** Navigate into the backend folder, open the docker-compose.yml file and input your mysql user and password details. Then open your terminal and run the following commands:
* docker-compose build
* docker-compose up -d
* docker-compose run app php artisan migrate