```r
library(shiny)
library(shinythemes)
library(jsonlite)
library(DT)
library(dplyr)
library(httr)
library(stringr)
library(tidytext)
library(ggplot2)
library(tidyr)
library(lubridate)
library(purrr)

# Genre ID to Name Mapping for apple IOS app store search for inital drop down
GLOBAL_GENRE_ID_TO_NAME_MAP <- c(
  "6000" = "Business", "6001" = "Weather", "6002" = "Utilities", "6003" = "Travel",
  "6004" = "Sports", "6005" = "Social Networking", "6006" = "Reference", "6007" =
"Productivity",
  "6008" = "Photo & Video", "6009" = "News", "6010" = "Navigation", "6011" = "Music",
  "6012" = "Lifestyle", "6013" = "Health & Fitness", "6014" = "Games", "6015" = "Finance",
  "6016" = "Entertainment", "6017" = "Education", "6018" = "Books", "6020" = "Medical",
  "6021" = "Magazines & Newspapers", "6022" = "Catalogs", "6023" = "Food & Drink",
  "6024" = "Shopping", "6025" = "Stickers", "6026" = "Developer Tools", "6027" = "Graphics &
Design"
)

# list of phrases to be used for heuristic analysis
GLOBAL_HEURISTIC_DEFINITIONS_LIST <- list(
  "Visibility of System Status" = c( "won't load", "loading", "sync", "lag", "delay", "status",
"showing the status", "no status", "status",
    "progress", "wait", "waiting", "slow", "stuck", "refresh", "closes",
    "feedback", "responsive", "unresponsive", "timeout", "save", "saving", "tracking", "syncing",
"synchronizing"
  ),
  "Match System & Real World" = c(
    "language", "term", "icon", "model","clear", "confusing", "understand", "jargon", "meaning",
"intuitive",
    "familiar", "expected", "unexpected", "natural", "makes sense", "weird"
  ),
  "User Control & Freedom" = c("undo", "exit", "cancel", "reverse", "can't save", "re-order",
"dosen't show", "re-do","redo", "mistake", "oops", "stuck", "trap", "forced", "escape", "close",
"won't let", "won't let you", "can't save",
    "delete", "remove", "change", "edit", "go back", "locked out", "can't remove", "can't exit",
"litteraly can't", "unable to",
    "not allowed", "not possible", "not able to", "not able", "can't do that", "can't do this", "can't
go back","not allow me", "won't allow me"
  ),
  "Consistency & Standards" = c("consistent", "standard", "reliable", "familiar","inconsistent",
"different", "layout", "placement", "looks different",
    "behaves differently", "predictable", "uniform", "pattern", "work properly"
  ),
  "Error Prevention" = c("bug", "crash", "error", "freeze", "delete", "lost", "issue", "broken",
    "fail", "glitch", "nag", "bait", "doesn't accecpt", "won't let me", "won't accept","broken",
"doesn't work", "not working", "warning", "confirm", "confirmation",
    "prevent", "avoid", "accidental", "lost data", "corrupt", "intrusive", "popup", "annoying",
"umable to"
  ),
  "Recognition Rather Than Recall" = c( "remember", "recall", "can't find", "search", "no
notification",
    "forgot", "memory", "locate", "navigate", "menu", "hidden", "obvious", "visible",
    "steps", "instructions", "easy to find", "hard to find", "lot of questions"
  ),
  "Flexibility & Efficiency" = c("shortcut", "custom", "advanced", "speed", "fast",
    "slow", "quick", "efficient", "inefficient", "customize", "personalize", "expert",
    "beginner", "steps", "tedious", "automate", "performance", "app freezes", "cannot update"
  ),
  "Aesthetic & Minimalist Design" = c("clutter", "ugly", "interface", "visual", "design", "buggy",
"frustraiting", " ui ", "confusing",
    "clean", "simple", "minimal", "layout", "busy", "confusing", "looks dated", "modern",
"unusable",
    "font", "color", "theme", "look", "feel", "style", "appealing", "attractive", "organized",
"distracting"
  ),
  "Help Users Recover" = c("recover", "fix", "help", "support", "guide", "error message",
"explain", "solution", "suggestion", "troubleshoot", "contact",
    "retry", "alert", "notification", "understand error", "what went wrong"
  ),
  "Help & Documentation" = c(
    "doc", "manual", "tutorial", "instruction", "hard to find", "impossible to find",
    "faq", "tips", "learn", "onboarding", "explain", "explanation", "help section",
    "example", "video", "getting started"
  )
)
# Function to help me go throgh every catagory for the analysis
GLOBAL_HEURISTIC_KEYWORDS_DF <-
bind_rows(lapply(names(GLOBAL_HEURISTIC_DEFINITIONS_LIST), function(h_name)
tibble(heuristic = h_name, keyword = GLOBAL_HEURISTIC_DEFINITIONS_LIST[[h_name]])))
GLOBAL_BING_SENTIMENT_LEXICON_DF <- get_sentiments("bing")
```

```
# app ui and custom html to make the app look better
ui <- fluidPage(
  theme = shinytheme("paper"),
  tags$head(
    tags$style(HTML("
      body { padding-top: 20px; }
      .shiny-title-panel h2 { margin-bottom: 25px; }
      .btn-default { margin-right: 5px; margin-bottom: 5px;}
      .selected-app-header {
         margin-bottom: 20px;
         padding: 15px;
         background-color: #f9f9f9;
         border-radius: 4px;
         border: 1px solid #eee;
         display: flex;
         align-items: center;
         justify-content: center;
      }
      .selected-app-header img, .selected-app-header .placeholder-icon {
         width: 40px;
         height: 40px;
         border-radius: 6px;
         margin-right: 15px;
         flex-shrink: 0;
      }
       .selected-app-header h4 {
         margin: 0;
         font-size: 1.1em;
         line-height: 1.2;
       }
      .analysis-prompt-card {
         text-align: center;
         padding: 20px;
         margin-top: 20px;}
      .results-summary-banner {
         font-size: 1.1em;
         padding: 10px;
         margin-bottom: 15px;
         border-radius: 4px; }
      .alert-info {
         background-color: #e3f2fd;
         color: #1e88e5;
```

```
        border-color: #bbdefb; }
      .alert-success {
        background-color: #e8f5e9;
        color: #388e3c;
        border-color: #c8e6c9; }
      .alert-warning {
        background-color: #fff3e0;
        color: #f57c00;
        border-color: #ffe0b2; }
      .help-text { color: #757575; font-size: 0.9em; }
      mark {
       background-color: #FFFF99;
       padding: 0.2em; }
    ")),
    tags$script(HTML("
      $(document).on('keypress', '#search_term_input_field', function(e) {
        if (e.which == 13) { e.preventDefault(); $('#search_button').click(); }
      });
    "))
  ),

  titlePanel("App Review UX Heuristic Analyzer"),
  uiOutput("selected_app_header_ui"),
  hr(style="margin-top: 0; margin-bottom: 20px;"),
  uiOutput("main_content_area_ui")
)



# Server function for how the app will search apps / run analysis based on selection ect
server <- function(input, output, session) {

# creating special objects for my app so that it can change during the apps execustion
  app_data <- reactiveValues(
    search_results_df = NULL,
    search_validation_message = NULL,
    selected_app_id = NULL,
    selected_app_name = NULL,
    selected_app_icon_url = NULL,
    raw_review_feed_df = NULL,
    process_log_text = "",
    analysis_data_ready_flag = FALSE,
    cleaned_reviews_df = NULL,
    heuristic_plot_data_df = NULL,
    current_ui_view = "app_search",
    last_search_query_text = "",
    last_search_genre_id = "N/A"
  )

  analysis_state <- reactiveValues(
    active_heuristic_name = NULL
  )


  `%||%` <- function(value, default_scalar) {
    if (is.null(value) || length(value) == 0 || all(is.na(value)) || (is.character(value) &&
all(nchar(trimws(value)) == 0))) {
      return(default_scalar)
    }
    return(value)
  }

  format_large_number <- function(number_value) {
    if (is.na(number_value) || !is.numeric(number_value)) return("N/A")
    if (number_value < 1000) return(as.character(number_value))
    if (number_value < 1000000) return(paste0(round(number_value/1000,
ifelse(number_value %% 1000 == 0, 0, 1)), "K"))
    return(paste0(round(number_value/1000000, 1), "M"))
  }


  # Have the user first search for an app in the app store and show a list of apps
  execute_app_search <- function() {
    search_query_raw <- trimws(input$search_term_input_field)
    search_query_for_api <- gsub(" +", "+", search_query_raw)

    if (nchar(search_query_raw) == 0 && input$genre_filter_input == "N/A") {
      app_data$search_validation_message <- "Please enter a search term or select a Genre."
      return(NULL)
    }
    if (nchar(search_query_raw) > 100) {
      app_data$search_validation_message <- "Search term must be 100 characters or less."
      return(NULL)
    }
    app_data$search_validation_message <- NULL

    api_search_term <- ifelse(nchar(search_query_for_api) > 0, search_query_for_api, "")
    genre_id_for_api <- if (input$genre_filter_input != "N/A") input$genre_filter_input else NA
    itunes_search_api_url <- paste0("https://itunes.apple.com/search?term=",
api_search_term,
                        "&country=US&media=software&entity=software&limit=", 50)
    if (!is.na(genre_id_for_api)) itunes_search_api_url <- paste0(itunes_search_api_url,
"&genreId=", genre_id_for_api)

    api_response_json <- tryCatch(jsonlite::fromJSON(itunes_search_api_url), error =
function(e) {
      app_data$search_validation_message <- paste("Error fetching app list:", e$message);
NULL })

    if (is.null(api_response_json) || length(api_response_json$results) == 0) {
      app_data$search_validation_message <- "No apps found for your search."
      return(NULL)
    }

    apps_from_api <- api_response_json$results
    data.frame(
      AppIconUrl = apps_from_api$artworkUrl60 %||% NA_character_,
      AppName = apps_from_api$trackCensoredName %||% "N/A",
      AppTrackId = apps_from_api$trackId %||% NA_character_,
      AppGenre = apps_from_api$primaryGenreName %||% "N/A",
      AverageRating = if("averageUserRating" %in% names(apps_from_api))
apps_from_api$averageUserRating else NA_real_,
      NumberOfReviews = if("userRatingCount" %in% names(apps_from_api))
apps_from_api$userRatingCount else NA_integer_,
      OriginalAppName = apps_from_api$trackCensoredName %||% "N/A",
      OriginalIconUrl = apps_from_api$artworkUrl60 %||% NA_character_,
      stringsAsFactors = FALSE
    )
  }

  # UI for the search for app function
  output$main_content_area_ui <- renderUI({
    current_display_view <- app_data$current_ui_view

    if (current_display_view == "app_search") {
      fluidRow(
       column(width = 4,
           wellPanel(
            h4("Search for App"),
            textInput("search_term_input_field", "Search Term:",
                value = isolate(app_data$last_search_query_text %||% ""), placeholder = "e.g.,
puzzle, notes, fitness"),
           selectInput("genre_filter_input", "Genre:",
                 choices = c("All Genres" = "N/A",
setNames(names(GLOBAL_GENRE_ID_TO_NAME_MAP),
unname(GLOBAL_GENRE_ID_TO_NAME_MAP))),
                 selected = isolate(app_data$last_search_genre_id %||% "N/A")),
           actionButton("search_button", "Search Apps", class = "btn-primary", icon =
icon("search")),
           actionButton("reset_search_fields_button", "Start Over", class = "btn-default", icon
= icon("refresh")),
           textOutput("search_validation_message_display")
          )
      ),
      column(width = 8,
          if (!is.null(app_data$search_results_df)) {
           div(
            p(class="alert alert-info", icon("lightbulb", class="fa-regular"), " Select an app
below to analyze its reviews."),
            h4("Search Results"),
            tags$p(if (nrow(app_data$search_results_df) > 0)
paste(nrow(app_data$search_results_df), "apps found") else "0 apps found"),
            DT::dataTableOutput("app_search_results_table")
           )
          } else if (!is.null(app_data$search_validation_message) &&
                app_data$search_validation_message != "Please enter a search term or select
a Genre." &&
                app_data$search_validation_message != "Search term must be 100
characters or less.") {
           div(
            p(class="alert alert-info", icon("lightbulb", class="fa-regular"), " Enter search
criteria to find apps."),
            h4("Search Results"),
            tags$p(app_data$search_validation_message),
            DT::dataTableOutput("app_search_results_table")
           )
          }
      )
     )
   } else if (current_display_view == "analysis_setup_prompt") {
     div(class="container-fluid analysis-prompt-card",
       wellPanel(
        h4(paste("Ready to Analyze:", app_data$selected_app_name %||% "Selected App")),
        p("This process will fetch the latest reviews and perform UX heuristic analysis. It may
take a few moments."), br(),
        actionButton("initiate_full_analysis_button", "Fetch & Analyze Reviews", class = "btn-
success btn-lg", icon = icon("cogs")),
        br(),br(),
        actionButton("navigate_to_app_search_button", "Back to App Search", class = "btn-
default")
       )
     )
   } else if (current_display_view == "analysis_inprogress") {
     div(class="container-fluid", style="text-align:center; padding: 20px;",
       h4(icon("spinner", class="fa-spin"), " Analyzing reviews... please wait.")
     )
   } else if (current_display_view == "analysis_results_view") {
     tagList(
      actionButton("navigate_to_search_from_results_button", "Analyze Another App",
class="btn-default", icon=icon("arrow-left"), style="margin-bottom:15px; margin-
left:15px;"),
      div(class="container-fluid",
        if(!is.null(app_data$cleaned_reviews_df) && nrow(app_data$cleaned_reviews_df) >
0){
          tags$div(class="alert alert-success results-summary-banner",
paste(nrow(app_data$cleaned_reviews_df), "reviews processed for analysis."))
        } else if (!is.null(app_data$raw_review_feed_df) &&
nrow(app_data$raw_review_feed_df) == 0) {
          tags$div(class="alert alert-warning results-summary-banner", "No reviews were
found or processed for this app.")
        },
        tabsetPanel(
          id = "analysis_results_tabs_panel",
          tabPanel("UX Heuristic Analysis", value = "ux_analysis_tab", br(),
               sidebarLayout(
                sidebarPanel(width = 4,
                       dateRangeInput("filter_date_range_input", "Filter by Review Date:",
                              start = Sys.Date() - 365, end = Sys.Date(), format="yyyy-mm-dd"),
                       selectInput("filter_version_select_input", "Filter by App Version:", choices
= c("All"), selected = "All"),
                       hr(), DTOutput("heuristic_summary_display_table"), br(),
                       p(class="help-text", "Click a row in the table above or a bar in the plot to
see detailed quotes.")
                ),
                mainPanel(width = 8,
                     tabsetPanel( id = "ux_details_tabs_panel",
                            tabPanel("Issue Frequency Plot", value = "ux_frequency_plot_tab",
                                        plotOutput("ux_heuristic_frequency_plot", height = "550px",
click = "plot_interaction_click_handler")
                                   ),
                                   tabPanel("Issue Details", value = "ux_quotes_detail_tab",
                                        h4(textOutput("selected_heuristic_label_display")),
                                        DTOutput("heuristic_quotes_display_table")
                                   )
                             )
                        )
                      )
             ),
             tabPanel("Fetched Raw Reviews", value="raw_review_data_tab", br(),
                  wellPanel(h5("Sample of Fetched Reviews (Max 500)"),
DT::dataTableOutput("raw_reviews_display_table"))
             )
            )
          )
        )
      }
    })

    # looks for what row the user selects for the app's list to analyze
    observeEvent(input$app_search_results_table_rows_selected, {
      selected_row_idx <- input$app_search_results_table_rows_selected
      req(app_data$search_results_df, length(selected_row_idx) > 0, selected_row_idx <=
nrow(app_data$search_results_df))

      chosen_app_details <- app_data$search_results_df[selected_row_idx, ]
      app_data$selected_app_id <- chosen_app_details$AppTrackId
      app_data$selected_app_name <- chosen_app_details$OriginalAppName
      app_data$selected_app_icon_url <- chosen_app_details$OriginalIconUrl

      app_data$raw_review_feed_df <- NULL; app_data$cleaned_reviews_df <- NULL;
      app_data$analysis_data_ready_flag <- FALSE; analysis_state$active_heuristic_name <-
NULL;
      app_data$current_ui_view <- "analysis_setup_prompt"
    })

    # back to search button to let the user go back
    generic_navigate_to_search_view <- function() {
      app_data$current_ui_view <- "app_search"
      app_data$raw_review_feed_df <- NULL
      app_data$cleaned_reviews_df <- NULL
      analysis_state$active_heuristic_name <- NULL
```

```
 }
 observeEvent(input$navigate_to_app_search_button, {
generic_navigate_to_search_view() })
 observeEvent(input$navigate_to_search_from_results_button, {
generic_navigate_to_search_view() })

 # search query button to search for the app
 observeEvent(input$search_button, {
  app_data$last_search_query_text <- input$search_term_input_field
  app_data$last_search_genre_id <- input$genre_filter_input
  app_data$search_results_df <- NULL
  app_data$search_validation_message <- NULL

  notification_id <- showNotification("Searching iTunes...", duration = NULL, closeButton =
FALSE, type = "message")
  on.exit(removeNotification(notification_id), add = TRUE)

  search_outcome_df <- execute_app_search()
  if (!is.null(search_outcome_df)) app_data$search_results_df <- search_outcome_df else
app_data$search_results_df <- NULL
 })

 # reset the search button
 observeEvent(input$reset_search_fields_button, {
  updateTextInput(session, "search_term_input_field", value = "")
  updateSelectInput(session, "genre_filter_input", selected = "N/A")
  app_data$last_search_query_text <- ""
  app_data$last_search_genre_id <- "N/A"
  app_data$search_results_df <- NULL; app_data$search_validation_message <- NULL;
app_data$selected_app_id <- NULL;
  app_data$selected_app_name <- NULL; app_data$selected_app_icon_url <- NULL;
app_data$raw_review_feed_df <- NULL;
  app_data$process_log_text <- ""; app_data$analysis_data_ready_flag <- FALSE;
  app_data$cleaned_reviews_df <- NULL; analysis_state$active_heuristic_name <- NULL
  app_data$current_ui_view <- "app_search"
 })

 # Search validation message
 output$search_validation_message_display <- renderText({
app_data$search_validation_message })

 # Table output once the term is serarched
 output$app_search_results_table <- DT::renderDataTable({
  results_df_for_display <- if (is.null(app_data$search_results_df)) {
     data.frame(Icon = character(), Name = character(), Genre = character(), Rating =
character(), Reviews = character(), stringsAsFactors = FALSE)
    } else {
     app_data$search_results_df %>%
      mutate(
       IconHtml = paste0("<img src='", AppIconUrl, "' style='width:40px;height:40px;vertical-
align:middle; border-radius: 6px;' onerror='this.style.display=\"none\"'>"),
       AppNameDisplay = AppName,
       ReviewsFormatted = sapply(NumberOfReviews, format_large_number),
       RatingDisplay = ifelse(is.na(AverageRating), "N/A", sprintf("%.1f", AverageRating))
      ) %>%
      select("App Name" = IconHtml, " " = AppNameDisplay, Rating = RatingDisplay, Reviews
= ReviewsFormatted)
    }

    DT::datatable(results_df_for_display, escape = FALSE, selection = "single", rownames =
FALSE,
             class = 'cell-border stripe compact hover',
             options = list(pageLength = 10, autoWidth = FALSE, scrollX = FALSE, dom = 'rtip',
                       columnDefs = list(
                        list(width = '60px', targets = 0, orderable = FALSE, searchable = FALSE,
className = 'dt-center'),
                        list(width = '60%', targets = 1),
                        list(width = '15%', targets = 2, className = 'dt-center'),
                        list(width = '15%', targets = 3, className = 'dt-center')
                       ),
                       language = list(emptyTable = "No apps found matching your criteria.",
                               zeroRecords = "No apps found matching your filter.")
             )
    )
  })

  # Logic to get the reviews for the selected app and to scrape all of the reviews from the
past 500 comments
  fetch_all_review_data <- function(target_app_id, target_app_name) {
    all_review_pages_dfs <- list()
    max_review_pages <- 10; fetch_was_successful <- TRUE
    for (page_index in 1:max_review_pages) {
     current_progress_value <- 0.05 + ((page_index / max_review_pages) * 0.40)
     shiny::setProgress(value = current_progress_value, detail = paste("Fetching review
page", page_index))

     reviews_rss_url <- paste0("https://itunes.apple.com/us/rss/customerreviews/page=",
page_index, "/id=", target_app_id, "/sortBy=mostRecent/json")
```

```
    http_response <- tryCatch(httr::GET(reviews_rss_url, httr::timeout(10)), error =
function(e) {
      fetch_was_successful <<- FALSE; NULL })
    if (!fetch_was_successful || is.null(http_response) || httr::status_code(http_response) !=
200) {
      fetch_was_successful <<- FALSE; break }

    review_page_json_text <- httr::content(http_response, as = "text", encoding = "UTF-8")
    review_page_data_list <- tryCatch(jsonlite::fromJSON(review_page_json_text, flatten =
TRUE), error = function(e) {
      fetch_was_successful <<- FALSE; NULL })

    if (!fetch_was_successful || is.null(review_page_data_list$feed) ||
is.null(review_page_data_list$feed$entry) || NROW(review_page_data_list$feed$entry) ==
0) {
      if (page_index == 1) {
        fetch_was_successful <<- FALSE;
      } else {
      }
      break
    }

    current_page_reviews_df <- as.data.frame(review_page_data_list$feed$entry,
stringsAsFactors = FALSE)
    if (page_index == 1 && nrow(current_page_reviews_df) > 0) {
      first_review_entry <- current_page_reviews_df[1, , drop = FALSE]
      is_metadata_entry <- any(sapply(c("im:rating.label", "im.rating", "content.label",
"id.label"), function(col_name_to_check) {
        if (col_name_to_check %in% names(first_review_entry)) {
          value_in_col <- first_review_entry[[col_name_to_check]][1]
          return(is.na(value_in_col) || value_in_col == "" || (col_name_to_check == "id.label" &&
grepl(paste0("^", target_app_id, "$"), value_in_col)))
        }
        return(FALSE)
      }))
      if(is_metadata_entry) {
        current_page_reviews_df <- if (nrow(current_page_reviews_df) > 1)
current_page_reviews_df[-1, , drop = FALSE] else data.frame()
      }
    }

    if (nrow(current_page_reviews_df) > 0) {
      all_review_pages_dfs[[length(all_review_pages_dfs) + 1]] <- current_page_reviews_df
    } else {
      if (page_index == 1) {
        fetch_was_successful <<- FALSE;
      }
      break
    }
    Sys.sleep(0.25)
  }

  if (fetch_was_successful && length(all_review_pages_dfs) > 0) {
    combined_reviews_from_json <- dplyr::bind_rows(all_review_pages_dfs)
    num_raw_rows <- nrow(combined_reviews_from_json); flattened_review_list <- list()

    extract_column_vector <- function(input_df, column_name_options, expected_length) {
      for (col_opt in column_name_options) {
        if (col_opt %in% names(input_df)) {
          column_data_raw <- input_df[[col_opt]]
          if (is.data.frame(column_data_raw) && "label" %in% names(column_data_raw))
column_data_raw <- column_data_raw[["label"]]
          else if (is.list(column_data_raw)) {
            column_data_raw <- sapply(column_data_raw, function(item) item[["label"]] %||%
(if(is.list(item) && length(item)>0) as.character(item[[1]]) else as.character(item)) %||%
NA_character_)
          }
          return(rep_len(as.character(column_data_raw), expected_length))
        }
      }
      return(rep(NA_character_, expected_length))
    }

    tryCatch({
      flattened_review_list$author_uri  <-
extract_column_vector(combined_reviews_from_json, c("author.uri.label", "author.uri"),
num_raw_rows)
      flattened_review_list$author_name <-
extract_column_vector(combined_reviews_from_json, c("author.name.label",
"author.name"), num_raw_rows)
      flattened_review_list$updated_timestamp <-
extract_column_vector(combined_reviews_from_json, c("updated.label", "updated"),
num_raw_rows)
      flattened_review_list$rating_value  <-
extract_column_vector(combined_reviews_from_json, c("im:rating.label", "im.rating"),
num_raw_rows)
```

```
    flattened_review_list$app_version   <-
extract_column_vector(combined_reviews_from_json, c("im:version.label", "im.version"),
num_raw_rows)
    flattened_review_list$review_id    <-
extract_column_vector(combined_reviews_from_json, c("id.label", "id"), num_raw_rows)
    flattened_review_list$review_title  <-
extract_column_vector(combined_reviews_from_json, c("title.label", "title"),
num_raw_rows)
    flattened_review_list$review_content<-
extract_column_vector(combined_reviews_from_json, c("content.label", "content"),
num_raw_rows)
    flattened_review_list$vote_sum     <-
extract_column_vector(combined_reviews_from_json, c("im:voteSum.label",
"im.voteSum"), num_raw_rows)
    flattened_review_list$vote_count   <-
extract_column_vector(combined_reviews_from_json, c("im:voteCount.label",
"im.voteCount"), num_raw_rows)

    structured_reviews_df <- as.data.frame(flattened_review_list, stringsAsFactors =
FALSE)
   }, error = function(e) {
    structured_reviews_df <- data.frame(); fetch_was_successful <<- FALSE })

   if (fetch_was_successful && nrow(structured_reviews_df) > 0 && "review_id" %in%
names(structured_reviews_df) && "review_content" %in% names(structured_reviews_df)) {
    structured_reviews_df <- structured_reviews_df[!
(is.na(structured_reviews_df$review_id %||% NA) &
is.na(structured_reviews_df$review_content %||% NA)), ]
   }
   app_data$raw_review_feed_df <- structured_reviews_df

  } else if (fetch_was_successful) {
   app_data$raw_review_feed_df <- tibble()
  } else {
   app_data$raw_review_feed_df <- NULL
  }
  return(fetch_was_successful && !is.null(app_data$raw_review_feed_df))
 }


 # Now the gathered data is cleaned and prepared for analysis
 prepare_data_for_ux_analysis <- function() {
  shiny::setProgress(value = 0.50, detail = "Cleaning review data...")
```

```
    temp_cleaned_reviews <- app_data$raw_review_feed_df %>%
      mutate(
        review_id = as.character(review_id %||% ""),
        updated_timestamp = as.character(updated_timestamp %||% ""),
        rating_value = as.character(rating_value %||% ""),
        app_version = as.character(app_version %||% ""),
        review_title = as.character(review_title %||% ""),
        review_content = as.character(review_content %||% ""),
        author_name = as.character(author_name %||% ""),
        vote_sum = as.character(vote_sum %||% "")
      ) %>%
      select(
        id = review_id,
        updated = updated_timestamp,
        rating = rating_value,
        version = app_version,
        title = review_title,
        review_comment_text = review_content,
        username = author_name,
        comment_votes_sum = vote_sum
      ) %>%
      mutate(
        cleaned_title = str_squish(str_to_lower(title)),
        cleaned_review_comment = str_squish(str_to_lower(review_comment_text)),
        full_review_text = if_else(cleaned_title == "" & cleaned_review_comment == "", "",
paste(cleaned_title, cleaned_review_comment, sep = ". ")),
        full_review_text = str_remove_all(full_review_text, "^\\.\\s*|\\s*\\.$|^\\.$"),
        review_date = lubridate::as_date(lubridate::ymd_hms(updated)),
        review_date = if_else(is.na(review_date), lubridate::as_date(str_sub(updated, 1, 10)),
review_date),
        version = as.character(version %||% "Unknown")
      ) %>%
      select(-cleaned_title, -cleaned_review_comment)

    shiny::setProgress(value = 0.90, detail = "Finalizing preparation...")
    app_data$cleaned_reviews_df <- temp_cleaned_reviews
    app_data$analysis_data_ready_flag <- TRUE
    return(TRUE)
  }

  # Header to show what app the user had selected
  output$selected_app_header_ui <- renderUI({
    if (!is.null(app_data$selected_app_name) && app_data$current_ui_view !=
"app_search") {
      div(class = "selected-app-header",
        if(!is.null(app_data$selected_app_icon_url)) {
          img(src=app_data$selected_app_icon_url)
        } else {
          div(class="placeholder-icon", style="background-color: #eee;")
        },
        h4(strong(app_data$selected_app_name %||% "App"))
      )
    } else { NULL }
  })


  # run full analysis
  observeEvent(input$initiate_full_analysis_button, {
    req(app_data$selected_app_id, app_data$selected_app_name)
    app_data$current_ui_view <- "analysis_inprogress"
    app_data$process_log_text <- ""

    withProgress(message = 'Starting Analysis...', value = 0, {
      shiny::setProgress(value = 0.05, message = 'Fetching App Reviews...', detail =
'Initializing...')
      fetch_completed_successfully <- fetch_all_review_data(target_app_id =
app_data$selected_app_id, target_app_name = app_data$selected_app_name)

      if (fetch_completed_successfully && !is.null(app_data$raw_review_feed_df) &&
nrow(app_data$raw_review_feed_df) > 0) {
        shiny::setProgress(value = 0.45, message = 'Preparing Data for Analysis...', detail =
'Cleaning review data...')
        preparation_completed_successfully <- prepare_data_for_ux_analysis()

        if (preparation_completed_successfully) {
          shiny::setProgress(value = 1, message = "Analysis Complete!", detail = "Loading
results...")
          Sys.sleep(0.5)
          app_data$current_ui_view <- "analysis_results_view"
          updateTabsetPanel(session, "analysis_results_tabs_panel", selected =
"ux_analysis_tab")
        } else {
          showNotification("Failed to prepare data for sentiment analysis.", type = "error",
duration = 7)
          app_data$current_ui_view <- "analysis_setup_prompt"
        }
      } else if (fetch_completed_successfully && !is.null(app_data$raw_review_feed_df) &&
nrow(app_data$raw_review_feed_df) == 0) {
        showNotification(paste0("No reviews found for '", app_data$selected_app_name,"'.
Cannot proceed with UX analysis."), type = "warning", duration = 7)
        app_data$cleaned_reviews_df <- tibble()
        app_data$analysis_data_ready_flag <- TRUE
        shiny::setProgress(value = 1, message = "No reviews found.", detail = "Displaying empty
results...")
        Sys.sleep(0.5)
        app_data$current_ui_view <- "analysis_results_view"
      } else {
        showNotification("Failed_to_fetch_app_reviews. Please try again or select another
app.", type = "error", duration = 7)
        app_data$current_ui_view <- "analysis_setup_prompt"
      }
    })
  })




  # update data based on filters selections
  observeEvent(app_data$cleaned_reviews_df, {
    req(app_data$cleaned_reviews_df)
    data_for_filters_update <- app_data$cleaned_reviews_df

    available_versions <- if (nrow(data_for_filters_update) > 0 && "version" %in%
names(data_for_filters_update)) na.omit(unique(data_for_filters_update$version)) else
character(0)
    version_choices_for_select <- if(length(available_versions) > 0) sort(available_versions)
else "N/A"
    current_version_filter_selection <- input$filter_version_select_input %||% "All"
    if (identical(version_choices_for_select, "N/A") && current_version_filter_selection !=
"All") current_version_filter_selection <- "All"
    else if (!(current_version_filter_selection %in% c("All", version_choices_for_select)))
current_version_filter_selection <- "All"
    updateSelectInput(session, "filter_version_select_input", choices = c("All",
version_choices_for_select), selected = current_version_filter_selection)

    available_dates <- if (nrow(data_for_filters_update) > 0 && "review_date" %in%
names(data_for_filters_update)) na.omit(data_for_filters_update$review_date) else
as.Date(character(0))
    min_date_from_data <- if(length(available_dates) > 0) min(available_dates, na.rm =
TRUE) else Sys.Date() - 365
    max_date_from_data <- if(length(available_dates) > 0) max(available_dates, na.rm =
TRUE) else Sys.Date()
```

```
    current_date_range_values <- input$filter_date_range_input
    updated_start_date <- if (!is.null(current_date_range_values) &&
!is.na(current_date_range_values[1]) && current_date_range_values[1] >=
min_date_from_data && current_date_range_values[1] <= max_date_from_data)
current_date_range_values[1] else min_date_from_data
    updated_end_date <- if (!is.null(current_date_range_values) &&
!is.na(current_date_range_values[2]) && current_date_range_values[2] <=
max_date_from_data && current_date_range_values[2] >= min_date_from_data)
current_date_range_values[2] else max_date_from_data
    if (is.finite(updated_start_date) && is.finite(updated_end_date) && updated_start_date >
updated_end_date) updated_start_date <- updated_end_date

    updateDateRangeInput(session, "filter_date_range_input", start = updated_start_date,
end = updated_end_date, min = min_date_from_data, max = max_date_from_data)
  })

  # Reactive Data, Filtered UX Data based on Inputs
  filtered_ux_analysis_data <- reactive({
    req(app_data$cleaned_reviews_df, app_data$analysis_data_ready_flag)
    base_reviews_for_ux <- app_data$cleaned_reviews_df
    if(nrow(base_reviews_for_ux) == 0) { return(tibble()) }

    reviews_with_full_text <- base_reviews_for_ux %>%
      filter(!is.na(full_review_text) & nchar(trimws(full_review_text)) > 0) %>%
      mutate(full_review_text = as.character(full_review_text), original_sentence_id =
row_number())
    if(nrow(reviews_with_full_text) == 0) return(tibble())

    tokenized_sentences_df <- tryCatch({
      reviews_with_full_text %>%
        select(id, version, original_review_text = full_review_text, review_date,
original_sentence_id) %>%
        unnest_tokens(output = "sentence_text", input = "original_review_text", token =
"sentences", drop = FALSE, to_lower = FALSE) %>%
        mutate(tokenized_sentence_unique_id = paste0(original_sentence_id, "_",
row_number()))
    }, error = function(e) tibble())
    if(nrow(tokenized_sentences_df) == 0) return(tibble())

    sentences_tagged_with_heuristics <- tokenized_sentences_df %>% rowwise() %>%
      mutate(
        lowercase_sentence = str_to_lower(as.character(sentence_text)),
        detected_heuristics =
list(GLOBAL_HEURISTIC_KEYWORDS_DF$heuristic[sapply(GLOBAL_HEURISTIC_KEYWOR
DS_DF$keyword, function(kw) str_detect(lowercase_sentence, fixed(kw)))])
      ) %>%
      ungroup() %>%
      mutate(detected_heuristics = ifelse(sapply(detected_heuristics, length) == 0,
list("Other"), detected_heuristics)) %>%
      unnest(cols = c(detected_heuristics)) %>% rename(heuristic_category =
detected_heuristics) %>%
      select(-lowercase_sentence)
    if(nrow(sentences_tagged_with_heuristics) == 0) return(tibble())

    sentences_with_sentiment_scores <- sentences_tagged_with_heuristics %>%
      mutate(sentence_for_sentiment_lib = str_to_lower(as.character(sentence_text))) %>%
      unnest_tokens(word, sentence_for_sentiment_lib, drop = FALSE) %>%
      inner_join(GLOBAL_BING_SENTIMENT_LEXICON_DF, by = "word", relationship = "many-
to-many") %>%
      mutate(sentiment_value = ifelse(sentiment == "negative", -1, 1)) %>%
      group_by(id, version, sentence_text, review_date, heuristic_category,
original_sentence_id, tokenized_sentence_unique_id, original_review_text) %>%
      summarise(overall_sentiment_score = sum(sentiment_value), .groups = "drop")

    final_ux_data_for_filtering <- sentences_tagged_with_heuristics %>%
      left_join(sentences_with_sentiment_scores, by = c("id", "version", "sentence_text",
"review_date", "heuristic_category", "original_sentence_id",
"tokenized_sentence_unique_id", "original_review_text")) %>%
      mutate(
        overall_sentiment_score = replace_na(overall_sentiment_score, 0),
        issue_severity_level = case_when(
          overall_sentiment_score <= -3 ~ "High",
          overall_sentiment_score < 0   ~ "Medium",
          TRUE                          ~ "Low"
        )
      ) %>%
      distinct(tokenized_sentence_unique_id, heuristic_category, .keep_all = TRUE)

    data_to_be_filtered <- final_ux_data_for_filtering
    selected_date_range <- input$filter_date_range_input
    if (!is.null(selected_date_range) && all(!is.na(selected_date_range)) &&
nrow(data_to_be_filtered) > 0 && "review_date" %in% names(data_to_be_filtered)) {
      data_to_be_filtered <- filter(data_to_be_filtered, !is.na(review_date) &
as.Date(review_date) >= as.Date(selected_date_range[1]) & as.Date(review_date) <=
as.Date(selected_date_range[2]))
    }
```

```
    selected_app_version <- input$filter_version_select_input
    if (!is.null(selected_app_version) && selected_app_version != "All" &&
nrow(data_to_be_filtered) > 0 && "version" %in% names(data_to_be_filtered)) {
      data_to_be_filtered <- filter(data_to_be_filtered, as.character(version) ==
as.character(selected_app_version))
    }
    return(data_to_be_filtered)
  })

  summarized_ux_heuristics_data <- reactive({
    data_for_ux_summary <- filtered_ux_analysis_data()
    if(is.null(data_for_ux_summary) || nrow(data_for_ux_summary) == 0) {
      return(tibble(heuristic_category = character(), MentionCount = integer(), PriorityScore =
numeric()))
    }
    data_for_ux_summary %>%
      group_by(heuristic_category) %>%
      summarise(
        MentionCount = n(),
        PriorityScore = sum(ifelse(overall_sentiment_score < 0, abs(overall_sentiment_score),
0)),
        .groups = "drop"
      ) %>%
      arrange(desc(PriorityScore))
  })


  # UX Heuristic Summary Table
  output$heuristic_summary_display_table <- renderDT({
    ux_summary_table_data <- summarized_ux_heuristics_data()
    req(ux_summary_table_data)

    app_data$heuristic_plot_data_df <- ux_summary_table_data %>%
      filter(heuristic_category != "Other" & MentionCount > 0) %>%
      arrange(MentionCount)

    datatable(ux_summary_table_data %>% select(Heuristic = heuristic_category, Score =
PriorityScore),
            selection = "single",
            rownames = FALSE,
            options = list(
              paging = FALSE,
              dom = 't',
          ordering = FALSE,
          columnDefs = list(
            list(width = '70%', targets = 0),
            list(width = '30%', targets = 1)
          )
        )
    )
  })

  observeEvent(input$heuristic_summary_display_table_rows_selected, {
    selected_table_row <- input$heuristic_summary_display_table_rows_selected
    current_summary_data <- summarized_ux_heuristics_data()
    if (length(selected_table_row) && !is.null(current_summary_data) &&
nrow(current_summary_data) >= selected_table_row) {
      chosen_heuristic <- current_summary_data$heuristic_category[selected_table_row]
      analysis_state$active_heuristic_name <- chosen_heuristic
      updateTabsetPanel(session, "ux_details_tabs_panel", selected =
"ux_quotes_detail_tab")
    }
  })

 # freqeuncy plot
  observeEvent(input$plot_interaction_click_handler, {
    plot_data_for_interaction <- app_data$heuristic_plot_data_df
    req(plot_data_for_interaction, nrow(plot_data_for_interaction) > 0)

    clicked_y_coordinate <- round(input$plot_interaction_click_handler$y)
    if (!is.null(clicked_y_coordinate) && clicked_y_coordinate >= 1 && clicked_y_coordinate
<= nrow(plot_data_for_interaction)) {
      chosen_heuristic_from_plot <-
plot_data_for_interaction$heuristic_category[clicked_y_coordinate]
      analysis_state$active_heuristic_name <- chosen_heuristic_from_plot
      updateTabsetPanel(session, "ux_details_tabs_panel", selected =
"ux_quotes_detail_tab")

      full_summary_table_data <- summarized_ux_heuristics_data()
      row_index_in_table <- which(full_summary_table_data$heuristic_category ==
chosen_heuristic_from_plot)
      if(length(row_index_in_table) > 0)
DT::selectRows(dataTableProxy("heuristic_summary_display_table"), selected =
row_index_in_table[1])
    }
  })
  output$selected_heuristic_label_display <- renderText({
    current_heuristic <- analysis_state$active_heuristic_name
    if (is.null(current_heuristic)) "Click a heuristic in the summary table or frequency plot to
view example quotes."
    else paste("Heuristic:", current_heuristic)
  })

  # table to show comments related to the heuristic selected and highlighted words
  output$heuristic_quotes_display_table <- renderDT({
    active_heuristic <- analysis_state$active_heuristic_name
    req(active_heuristic)
    quotes_data_source <- filtered_ux_analysis_data()
    req(quotes_data_source)

    keywords_to_highlight <- GLOBAL_HEURISTIC_KEYWORDS_DF %>%
      filter(heuristic == active_heuristic) %>%
      pull(keyword)

    quotes_for_selected_heuristic <- quotes_data_source %>%
      filter(heuristic_category == active_heuristic) %>%
      select(Date = review_date, Version = version, Sentence = sentence_text, Severity =
issue_severity_level) %>% # Use sentence_text
      distinct(Sentence, .keep_all = TRUE) %>%
      mutate(
        Sentence = purrr::reduce(keywords_to_highlight, function(text, kw) {
          stringr::str_replace_all(text, fixed(kw, ignore_case = TRUE), ~ paste0("<mark>", .x,
"</mark>"))
        }, .init = Sentence)
      ) %>%
      head(500)

    datatable(quotes_for_selected_heuristic,
              escape = FALSE,
              rownames = FALSE,
              options = list(pageLength = 15, dom = 'frtip', scrollY = '700px',
                             columnDefs = list(
                               list(width = '70%', targets = 2),
                               list(targets = 2, className = "dt-left")
                             )
              )
    )
  })

 # Frequency table
 output$ux_heuristic_frequency_plot <- renderPlot({
  plot_source_data <- summarized_ux_heuristics_data()
  req(plot_source_data)

  data_for_actual_plot <- plot_source_data %>% filter(heuristic_category != "Other" &
MentionCount > 0)
  if (nrow(data_for_actual_plot) == 0) {
   p_empty <- ggplot() +
    annotate("text", x = 0.5, y = 0.5, label = "No data available for the selected filters.", size =
5, color="grey50") +
    theme_void() +
    labs(title = "Issue Mentions by Heuristic") +
    theme(plot.title = element_text(size = rel(1.4), face = "bold", hjust = 0.5, margin =
margin(b=15)))
   return(p_empty)
  }

  ggplot(data_for_actual_plot, aes(x = reorder(heuristic_category, MentionCount), y =
MentionCount, fill = PriorityScore)) +
   geom_col(width = 0.7) + coord_flip() +
   scale_fill_viridis_c(option = "viridis", direction = -1, name = "Priority\nScore") +
   labs(x = NULL, y = "Number of Mentions", title = "Issue Mentions by Heuristic") +
   theme_minimal(base_size = 11) +
   theme(plot.title = element_text(size = rel(1.3), face = "bold", hjust = 0.5, margin =
margin(b=15)),
      axis.text = element_text(size = rel(0.9)), panel.grid.major.y = element_blank())
 }, height = 550)

 # All reviews used from the scrape for the user to be able to see
 output$raw_reviews_display_table <- DT::renderDataTable({
  req(app_data$raw_review_feed_df)
  if (nrow(app_data$raw_review_feed_df) == 0) {
   return(datatable(tibble(Message = "No raw reviews were fetched or available."),
rownames=FALSE, options=list(dom='t')))
  }

  columns_to_display_in_raw_table <- intersect(
   c("author_name", "updated_timestamp", "rating_value", "app_version", "review_title",
"review_content", "vote_sum", "vote_count"),
   names(app_data$raw_review_feed_df)
  )

  DT::datatable(head(app_data$raw_review_feed_df[, columns_to_display_in_raw_table,
drop=FALSE], 500),
                options = list(scrollX = TRUE, pageLength = 10, autoWidth = TRUE, dom = 'frtip')
   )
  })

}

shinyApp(ui, server)
```