```
# app.R
library(shiny)
library(bslib)
library(dplyr)
library(highcharter)
library(jsonlite)
library(reactable)
library(DBI)
library(RPostgres)
library(pool)



recursive_unname <- function(x) {
  if (is.list(x)) {
    lapply(x, recursive_unname)
  } else if (is.atomic(x)) {
    unname(x)
  } else {
    x
  }
}



# Helper function Max points from JSON
json_cache <- new.env(parent = emptyenv())
get_max_points <- function(json_str) {
  if (is.null(json_str) || is.na(json_str) || json_str == "" ||
json_str == "null") return(NA)
  json_str <- as.character(json_str)

  if (exists(json_str, envir = json_cache, inherits = FALSE)) {
    return(get(json_str, envir = json_cache))
  }
  parsed <- tryCatch(fromJSON(json_str), error = function(e)
NULL)

  if (is.null(parsed) || !(is.list(parsed) ||
is.data.frame(parsed)) || !("point_value" %in% names(parsed)) ||
length(parsed$point_value) == 0) {
    assign(json_str, NA, envir = json_cache)
    return(NA)
  }
  point_values_numeric <-
suppressWarnings(as.numeric(parsed$point_value))
```

```
  valid_points <-
point_values_numeric[!is.na(point_values_numeric)]

  if (length(valid_points) == 0) {
    assign(json_str, NA, envir = json_cache)
    return(NA)
  }
  m <- max(valid_points, na.rm = TRUE)
  res <- if (is.infinite(m)) NA else m
  assign(json_str, res, envir = json_cache)
  res
}





# Define the UI
ui <- fluidPage(
  theme = bs_theme(
    bootswatch    = "flatly",
    base_font     = font_google("Lato"),
    heading_font  = font_google("Lato")
  ),
  titlePanel("Evaluation Breakdown"),
  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId  = "selectedConference",
        label    = "Select a Conference:",
        choices  = c("All" = "All"),
        selected = "All"
      ),
      br(),
      conditionalPanel(
        condition = "input.selectedConference != 'All'",
        selectInput(
          inputId  = "selectedEvent",
          label    = "Select a Competitive Event:",
          choices  = c("All" = "All"),
          selected = "All"
        )
      ),
      conditionalPanel(
        condition = "input.selectedConference != 'All' &&
input.selectedEvent != 'All'",
        div(
          style = "margin-top:20px;",
          h4("Event Summary"),
          textOutput("overallAvgScore"),
          textOutput("totalEvaluations"),
          h5("Judges:"),
          uiOutput("judgeInfo")
        )
      )
    ),
    mainPanel(
      conditionalPanel(
        condition = "input.selectedConference != 'All'",
        highchartOutput("breakdownChart", height = "600px"),
        br(),
        reactableOutput("breakdownTable")
      )
    )
  )
)




# Server Logic

server <- function(input, output, session) {
  db_pool <- dbPool(
    drv      = RPostgres::Postgres(),
    dbname   = "tpsa",
    host     = "tramway.proxy.rlwy.net",
    port     = 11814,
    user     = "postgres",
    password = "pOCtIsjAssZfoeqzcaNLEgEHZCQFXeBf"
  )
  session$onSessionEnded(function() poolClose(db_pool))



  # 1) Conferences SQL Query
  conference_query <- "
    SELECT id, internal_name, start_date
      FROM conferences
     WHERE start_date IS NOT NULL
       AND start_date >= '2024-01-01'
       AND start_date <= '2025-12-31'
    ORDER BY start_date DESC;
  "
  conferenceData <- reactive({
    dbGetQuery(db_pool, conference_query)
  })
  observe({
    df <- conferenceData()
    choices <- c("All" = "All")
    if (nrow(df) > 0) {
      conf_choices <- setNames(df$id, paste0(df$internal_name, "
(", format(as.Date(df$start_date), "%Y"), ")"))
      choices <- c(choices, conf_choices)
      default_row_2025 <- df %>% filter(internal_name == "State
Conference", format(as.Date(start_date), "%Y") == "2025")
      default_row_any_state <- df %>% filter(internal_name ==
"State Conference")
      default_row_most_recent <- df %>% slice(1)
      default_id <- if (nrow(default_row_2025) == 1) {
        default_row_2025$id
      } else if (nrow(default_row_any_state) >= 1) {
        default_row_any_state %>% arrange(desc(start_date)) %>%
slice(1) %>% pull(id)
      } else if (nrow(default_row_most_recent) == 1) {
        default_row_most_recent$id
      } else {
      }

      updateSelectInput(session, "selectedConference", choices =
choices, selected = default_id)
      updateSelectInput(session, "selectedEvent", choices =
c("All" = "All"), selected = "All")
    } else {
      updateSelectInput(session, "selectedConference", choices =
choices, selected = "All") # Fallback if no confs
    }
  })


  # 2) Events for chosen conference (Filter Agility Events)
  eventsForConference <- eventReactive(input$selectedConference,
{
    req(input$selectedConference != "All")
    sql <- "
      SELECT DISTINCT cs.event_name
        FROM conference_schedule cs
       WHERE cs.conference = $1
         AND cs.competitive_event IS NOT NULL
       ORDER BY cs.event_name;
    "
    df <- dbGetQuery(db_pool, sql, params =
list(input$selectedConference))

    if (nrow(df) > 0) {
      df_filtered <- df[!grepl("agility", df$event_name,
ignore.case = TRUE), , drop = FALSE]
    } else {
      df_filtered <- df
    }

    c("All", df_filtered$event_name)
  })
  observeEvent(eventsForConference(), {
    current_selection <- input$selectedEvent
    valid_choices <- eventsForConference()
    if (is.null(current_selection) || !(current_selection %in%
valid_choices)) {
      current_selection <- "All"
    }
    updateSelectInput(session, "selectedEvent", choices =
valid_choices, selected = current_selection)
  }, ignoreNULL = FALSE)



  # 3) Fetching criteria details
  rawData <- reactive({
    req(input$selectedConference != "All")
    sql <- "
      SELECT
        e.score           AS evaluation_score,
        cs.event_name     AS competitive_event_name,
        ec.points         AS evaluation_criteria_points,
        crc.id            AS rubric_category_criteria_id,
        crc.name          AS rubric_category_criteria_name,
        crc.exemplary_options AS
rubric_category_criteria_exemplary_options,
        jcs.judges_id     AS joined_judge_id,
        jd.job_title      AS judge_job_title,
        ee.id             AS entry_id,
        e.id              AS evaluation_id
      FROM evaluation e
      JOIN evaluated_criteria ec
        ON e.id = ec.evaluationid
      JOIN competitive_event_rubric_criteria crc
        ON ec.criteriaid = crc.id
      JOIN event_entry ee
        ON e.entry = ee.id
      JOIN conference_schedule cs
        ON ee.event = cs.id
      LEFT JOIN judges_conference_schedule jcs
        ON cs.id = jcs.conference_schedule_id
      LEFT JOIN judges jd
        ON jcs.judges_id = jd.id
     WHERE cs.conference = $1
    "
    params <- list(input$selectedConference)
    if (input$selectedEvent != "All") {
      sql    <- paste0(sql, " AND cs.event_name = $2")
      params <- list(input$selectedConference,
input$selectedEvent)
    }

    fetched_data <- dbGetQuery(db_pool, sql, params = params)
    if ("rubric_category_criteria_exemplary_options" %in%
names(fetched_data)) {
      fetched_data$rubric_category_criteria_exemplary_options <-

as.character(fetched_data$rubric_category_criteria_exemplary_opt
ions)
    }
    return(fetched_data)
  })


  # 4) Aggregate criteria breakdown for a specific event

  aggData <- reactive({
    req(input$selectedConference != "All", input$selectedEvent
!= "All")
    df <- rawData()
    req(nrow(df) > 0)
    maxLookup <- df %>%
      distinct(rubric_category_criteria_id,
rubric_category_criteria_exemplary_options) %>%
      rowwise() %>%
      mutate(max_points =
get_max_points(rubric_category_criteria_exemplary_options)) %>%
      ungroup() %>%
      select(rubric_category_criteria_id, max_points) %>%
      filter(!is.na(max_points))

    # average awarded
    avgScores <- df %>%
      group_by(rubric_category_criteria_id,
rubric_category_criteria_name) %>%
      summarise(
        avg_criteria_points =
round(mean(evaluation_criteria_points, na.rm = TRUE), 2),
        n_scores           = n_distinct(evaluation_id), # Count
distinct evaluations
        .groups            = "drop"
      )


    # Join and calculate percentages, inner_join to keep only
those with valid max_points
    result <- avgScores %>%
      inner_join(maxLookup, by = "rubric_category_criteria_id")
%>%
      filter(!is.na(avg_criteria_points) & max_points > 0) %>%
      mutate(
        pct_achieved  = round(100 * avg_criteria_points /
max_points, 1),
        pct_remaining = pmax(0, 100 - pct_achieved)
      ) %>%
      arrange(desc(pct_achieved))

    if(nrow(result) == 0) {
      warning(glue::glue("aggData for event
'{input$selectedEvent}' yielded no rows after processing."))
      return(tibble(
        rubric_category_criteria_id = character(),
        rubric_category_criteria_name = character(),
        avg_criteria_points = numeric(),
        n_scores = integer(),
        max_points = numeric(),
        pct_achieved = numeric(),
        pct_remaining = numeric()
      ))
    } else {
      return(result)
    }
  })

  # 5) Aggregate all events when "All" is selected
  normalAggData <- reactive({
    req(input$selectedConference != "All", input$selectedEvent
== "All")
    sql <- "
```

```
        SELECT
            cs.event_name            AS competitive_event_name,
            ROUND(AVG(e.score), 1)  AS avg_event_score,
            COUNT(DISTINCT e.entry) AS n_evals
        FROM evaluation e
        JOIN event_entry ee ON e.entry = ee.id
        JOIN conference_schedule cs ON ee.event = cs.id
        WHERE
            cs.conference = $1
            AND cs.competitive_event IS NOT NULL
            AND cs.event_name NOT ILIKE '%agility%'
        GROUP BY cs.event_name
        HAVING COUNT(DISTINCT e.entry) > 0
        ORDER BY avg_event_score DESC;
    "
    dbGetQuery(db_pool, sql, params =
list(input$selectedConference))
  })



  # 8) Judge info list
  output$judgeInfo <- renderUI({
    req(input$selectedConference != "All", input$selectedEvent
!= "All")
    df <- rawData()
    req(nrow(df) > 0)

    judge_data <- df %>%
      filter(!is.na(joined_judge_id)) %>%
      group_by(joined_judge_id, judge_job_title) %>%
      # Count distinct evaluations associated with each judge
for this event
      summarise(n = n_distinct(evaluation_id), .groups = "drop")
%>%
      arrange(desc(n)) # Arrange by count

    if (nrow(judge_data) == 0) return(p("No judges found for
this event."))

    tagList(
      p(paste0("Number of Unique Judges: ",
n_distinct(judge_data$joined_judge_id))),
      tags$ul(
        lapply(seq_len(nrow(judge_data)), function(i) {
          # Provide fallback for missing job title
          display_title <-
ifelse(is.na(judge_data$judge_job_title[i]) |
judge_data$judge_job_title[i] == "",
                                 paste("Judge ID:",
judge_data$joined_judge_id[i]),
                                 judge_data$judge_job_title[i])
          tags$li(paste0(display_title, " (Evaluations: ",
judge_data$n[i], ")"))
        })
      )
    )
  })


  # 9) Unified chart
  output$breakdownChart <- renderHighchart({
    req(input$selectedConference != "All")

    if (input$selectedEvent == "All") {
      df <- normalAggData()
      req(nrow(df) > 0)

      color_vec <- ifelse(df$avg_event_score > 80, "#2ecc71",
                          ifelse(df$avg_event_score >= 60,
"#f1c40f", "#e74c3c"))
      event_data <- lapply(seq_len(nrow(df)), function(i) {
        list(
          y       = df$avg_event_score[i],
          color   = color_vec[i],
          custom  = list(
            event     = df$competitive_event_name[i],
            n_evals   = df$n_evals[i],
            avg_score = df$avg_event_score[i]
          )
        )
      }) %>% recursive_unname()

      highchart() %>%
        hc_chart(type = "bar") %>%
        hc_xAxis(categories = df$competitive_event_name,
                 title = list(text = "Competitive Events")) %>%
        hc_yAxis(min = 0, max = 100,
                 title = list(text = "Average Score (%)"),
                 labels = list(format = "{value}%")) %>%
        hc_add_series(name = "Average Score (%)", data =
event_data) %>%
        hc_plotOptions(bar = list(dataLabels = list(enabled =
FALSE))) %>%
        hc_title(text = "Competitive Event Breakdown", align =
"left") %>%
        hc_tooltip(useHTML = TRUE, formatter = JS("
          function() {
            var c = this.point.custom;
            // Check for null/NaN avg_score
            var avgScoreText = (c.avg_score === null ||
isNaN(c.avg_score)) ? 'N/A' : c.avg_score + '%';
            return '<b>' + c.event + '</b><br>' +
                   'Avg Score: ' + avgScoreText + '<br>' +
                   'Evaluations: ' + c.n_evals;
          }
        "))

    } else {
      df <- aggData()
      req(nrow(df) > 0)

      cols      <- ifelse(df$pct_achieved > 80, "#2ecc71",
                          ifelse(df$pct_achieved >= 60,
"#f1c40f", "#e74c3c"))
      achieved  <- lapply(seq_len(nrow(df)), function(i) list(
        y      = df$pct_achieved[i],
        color  = cols[i],
        custom = list(
          criterion = df$rubric_category_criteria_name[i],
          avg_score = df$avg_criteria_points[i],
          max_score = df$max_points[i],
          n_scores  = df$n_scores[i]
        )
      )) %>% recursive_unname()
      remaining <- lapply(df$pct_remaining, function(x) list(y =
x, color = "#dcdcdc")) %>% recursive_unname()

      highchart() %>%
        hc_chart(type = "bar") %>%
        hc_xAxis(categories = df$rubric_category_criteria_name,
                 title = list(text = "Criteria")) %>%
        hc_yAxis(min = 0, max = 100,
                 title = list(text = "Percent Achieved"),
                 labels = list(format = "{value}%")) %>%
        hc_add_series(name = "Achieved", data = achieved, stack
= "a") %>%
        hc_add_series(name = "Remaining", data = remaining,
stack = "a") %>%
        hc_plotOptions(bar = list(stacking = "normal")) %>%
        hc_title(text = paste("Criteria Breakdown for",
input$selectedEvent), align = "left") %>%
        hc_tooltip(useHTML = TRUE, formatter = JS("
          function() {
            if (this.series.name === 'Achieved') {
              var c = this.point.custom;
               var avgScoreFormatted = (c.avg_score === null ||
isNaN(c.avg_score)) ? 'N/A' :
Highcharts.numberFormat(c.avg_score, 1);
              return '<b>' + c.criterion + '</b><br/>' +
                     'Avg: ' + avgScoreFormatted + '/' +
c.max_score + '<br/>' +
                     'Evals: ' + c.n_scores;
            }
             return false; // Hide tooltip for 'Remaining'
segment
          }
        "))
    }
  })


  # 10) Unified table, normalAggData
  output$breakdownTable <- renderReactable({
    req(input$selectedConference != "All")

    if (input$selectedEvent == "All") {
      df <- normalAggData()
      req(nrow(df) > 0) # Add check
      reactable(
        df,
        columns = list(
          competitive_event_name = colDef(name = "Event"),
          avg_event_score        = colDef(name = "Avg Score
(%)", format = colFormat(suffix = "%")),
          n_evals                = colDef(name = "Evaluations")
        ),
        searchable      = TRUE,
        striped         = TRUE,
        highlight       = TRUE,
        paginationType  = "simple",
        defaultPageSize = 15
      )

    } else {
      df <- aggData()
      req(nrow(df) > 0)
      df_display <- df %>%
        select(
          Criteria = rubric_category_criteria_name,
          `Avg Score` = avg_criteria_points,
          `# Evaluations` = n_scores,
          `Max Points` = max_points,
          `Pct Achieved` = pct_achieved,
          `Pct Remaining` = pct_remaining
        )

      reactable(
        df_display,
        defaultSorted    = "Pct Achieved",
        defaultSortOrder = "desc",
        columns = list(
          Criteria        = colDef(minWidth = 200),
          `Avg Score`     = colDef(format = colFormat(digits =
2)),
          `# Evaluations` = colDef(),
          `Max Points`    = colDef(),
          `Pct Achieved`  = colDef(format = colFormat(suffix =
"%", digits = 1)),
          `Pct Remaining` = colDef(format = colFormat(suffix =
"%", digits = 1))
        ),
        searchable      = TRUE,
        striped         = TRUE,
        highlight       = TRUE,
        paginationType  = "simple",
        defaultPageSize = 15
      )
    }
  })
}

shinyApp(ui, server)
```