

Hogskolan Dalarna

# Building an interactive Web Application with Shiny Apps



DALARNA  
UNIVERSITY

AZZIMANI Yassine  
5-12-2024

# Introduction

This report outlines the development process of an interactive web application designed to display Points of Interest (POIs) based on user-selected municipalities in Sweden. The application leverages R Shiny, integrating various libraries and external API requests to enrich the user experience with dynamic content.

## Application Setup

### Initial Setup

The initial step involved setting up the basic structure for a Shiny application, which was divided into three main files to streamline development and maintain clarity:

**global.R:** For global settings including library dependencies.

**ui.R:** To design the user interface.

**server.R:** To handle server-side logic.

### Access and API Integration

**CeTLeR Lab API:** Ensuring access to the API and understanding its endpoints was crucial. This involved reading API documentation and possibly interacting with API endpoints via test scripts to understand the data schema and response format.

# Development

## Process

### Developing the User Interface (ui.R)

#### *Layout Design*

**Fluid Page Layout:** A fluid page layout was selected for its ability to adjust dynamically to different screen sizes. This approach ensures that the application remains usable and aesthetically pleasing across various devices.

#### *Interface Components*

**Map and Screen Real Estate:** The Leaflet map is carefully integrated into the interface to avoid dominating the entire screen. It shares the visual space with other UI components, allowing users to interact with the map without it overwhelming the other elements.

#### *Municipality Selection:*

**Dropdown Menu:** A dropdown menu was implemented to enable users to select from 20 major Swedish municipalities. This feature allows for straightforward navigation and selection, enhancing the user experience by making it easy to switch between different municipalities without cluttering the interface.

#### *POI Selection:*

**Radio Buttons:** Users can select the type of Point of Interest (POI) they wish to

view—Café, Restaurant, or Pub—via radio buttons. This method of selection simplifies the interface and provides a clear and concise choice without requiring additional navigation.

#### *Action Control:*

**Submit Button:** An "Import POIs" button was included to initiate the search and retrieval of POI data based on user selections. Its design ensures that the map and data table are updated seamlessly without needing to reload the page, enhancing the user experience.

**Component Placement and Styling:** The user interface components are arranged and styled to offer a harmonious and appealing appearance. The layout is crafted to guide users smoothly from selection through data visualization, ensuring the application is both effective and easy to use.

#### *Components:*

**Municipality Selection:** Implemented as a dropdown menu, allowing users to choose from a pre-defined list of major Swedish municipalities.

**POI Selection:** Radio buttons were used for selecting the type of POI, such as Cafés, Restaurants, and Pubs.

**Data Visualization:** A datatable was integrated to display the POIs' names and IDs, and a Leaflet map was added to geographically visualize the selected POIs.

## Implementing Server Logic (server.R)

#### *Data Fetching:*

A reactive event-driven model was used, where data fetching is triggered by user inputs.

Interaction with the CeTLer lab API was established via the `httr` library to retrieve POI data based on the selected municipality and POI type.

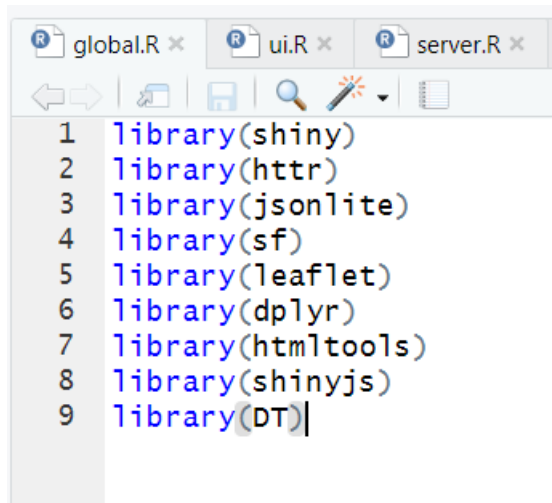
#### *Dynamic UI Updates:*

Reactive expressions and observer patterns were employed to update the data table and the map based on user inputs.

Temporary disabling of the fetch button post-interaction was implemented to prevent duplicate requests.

## Global Settings (global.R)

**Library Management:** Essential libraries such as `leaflet`, `httr`, `jsonlite`, and `dplyr` were loaded to facilitate API interactions, data processing, and mapping functionalities.



```
1 library(shiny)
2 library(httr)
3 library(jsonlite)
4 library(sf)
5 library(leaflet)
6 library(dplyr)
7 library(htmltools)
8 library(shinyjs)
9 library(DT)|
```

## Testing and Troubleshooting

**Local Testing:** The application underwent rigorous local testing to ensure that all interactive components were responsive and functioned as intended.

**Debugging:** Debugging was an iterative process, especially for handling asynchronous operations and API data fetching.

## Deployment

The final step involved deploying the fully functional application to a Shiny server, ensuring it was accessible and performed reliably in a live environment. During this phase, I encountered issues with missing libraries that were crucial for the application's operation. It became evident that certain required libraries had not been installed on the server, which prevented the application from

functioning correctly upon initial deployment.

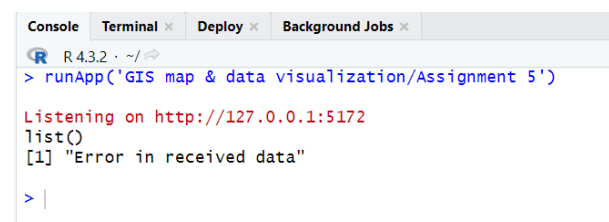
To resolve these issues, I meticulously checked the server environment to confirm the presence of all necessary R packages. I then installed any missing libraries and verified their compatibility with the server's existing software. After these adjustments, I redeployed the application, ensuring that all components were operational and that the application performed seamlessly in its intended environment.

This experience underscored the importance of thorough pre-deployment checks for software dependencies to prevent runtime issues and ensure a smooth user experience.

## Challenges Faced

### Asynchronous Data Handling

**Request Format Issues:** Initially, constructing API requests posed a significant challenge as the requests were not properly formatted, leading to errors and failed data retrievals.



```
R 4.3.2 · ~/
> runApp('GIS map & data visualization/Assignment 5')

Listening on http://127.0.0.1:5172
list()
[1] "Error in received data"
> |
```

**Multiple Rapid Clicks:** The UI encountered stability issues when the action button was clicked multiple times in quick succession. Implementing a solution to handle rapid, repeated interactions without crashing was essential.

### Variable Naming in Requests:

Discrepancies in variable names within the API requests caused problems in fetching the correct data. Aligning variable names in the code with those expected by the API was crucial to ensure accurate data retrieval and handling. These challenges required a combination of debugging, implementing checks and balances in the UI, and careful coordination with API documentation to ensure that all components interacted seamlessly.

## Debugging Process

During the development of the Shiny application, several challenges necessitated a thorough debugging process to ensure the application's functionality and user experience. Here's how I tackled these issues:

### Incorrect API Request Format

Initially, the application failed to retrieve data due to incorrectly formatted API requests. To resolve this, I carefully reviewed the API documentation to understand the expected parameters and data formats. Using logs and console outputs, I traced and identified mismatches in the request structure. I

corrected the query parameters and tested the changes locally to ensure that the API now responded with the correct data. This step was crucial in establishing a reliable data flow for the application.

```

})
response <- GET(url, query = params)
if (status_code(response) == 200) {
  json_data <- fromJSON(content(response, "text"))
  # Check if json_data is a list or a single object
  if (is.null(json_data) || !is.list(json_data)) {
    print(json_data)
    data <- st_as_sf(json_data,
                     crs = c("longitude", "latitude"), crs = 4326, agr = "constant")
    output$mapPlot <- renderTmap(
      tmap_mode("view")
    )
    tm_shape(data) + tm_symbols(size = 0.1, col = "blue")
  } else {
    print("Error in received data structure") # To debug if data is not ok
    output$mapPlot <- renderTmap(
      tmap_mode("view")
    )
    tm_shape() + tm_basemap(server = "OpenStreetMap")
  }
} else {
  print(paste("Error in API response:", status_code(response))) # Debug server error
  output$mapPlot <- renderTmap(
    tmap_mode("view")
  )
  tm_shape() + tm_basemap(server = "OpenStreetMap")
}
})
```

### Handling Rapid Successive Inputs

A significant issue arose when users clicked the "Import POIs" button multiple times quickly, which caused the application to freeze or crash. To debug this, I implemented a debouncing mechanism using shinyjs to disable the button immediately after a click and re-enable it only after the server had processed the previous request. This approach prevented multiple submissions and stabilized the application by controlling the flow of user inputs.

### In the server file:

```

  city = c(56.8777, 14.8091),
  "Växjö" = c(56.8777, 14.8091),
  "Halmstad" = c(56.6745, 12.8568),
  NULL) # NULL if no city found
}

server <- function(input, output) {
  observeEvent(input$go, {
    shinyjs::disable('go')
    shinyjs::delay(5000, shinyjs::enable('go'))
    if (is.null(input$city) || is.null(input$poiType)) {
      output$mapPlot <- renderTmap(
        tmap_mode("view")
      )
    }
  })
}
```

In the ui file:

```
ui <- fluidPage(  
  useShinyjs(),  
  titlePanel("UrbanSpotter - I  
  sidebarLayout(  
    sidebarPanel(  
      width = 1
```

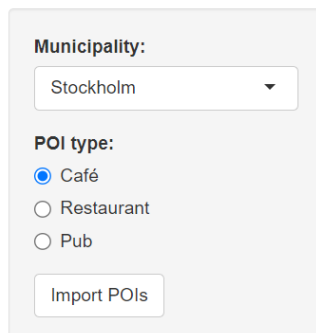
## Variable Naming Discrepancies

During the integration phase, discrepancies in variable names between the application's requests and the API's expectations led to errors. To pinpoint these issues, I used both server-side logging and client-side debugging tools to monitor what was being sent and received. After identifying the incorrect variable names, I refactored the relevant parts of the code to align with the API's requirements. This not only fixed the data fetching errors but also improved the overall reliability of data handling.

## Conclusion

This project not only enhanced my technical skills in R and Shiny but also improved my problem-solving abilities, particularly in the areas of API integration and dynamic content management. The deployment of the application provided a practical endpoint for this educational endeavor, demonstrating the application's functionality in a real-world scenario.

## Lab 5 - ShinyApps



Municipality:

Stockholm

POI type:

☒ Café

☐ Restaurant

☐ Pub

Import POIs

Error: objet 'lon' introuvable