

Package: cascadeR (via r-universe)

September 19, 2026

Title Shiny App to Explore Single-Cell Analysis

Version 0.99.0

Description Interactive and modular shiny app for single-cell and spatial transcriptomic data exploration and management. Visualize cell clusters or groups using interactive cell embeddings, e.g. UMAP or t-SNE, or spatial views. Explore cell composition and quality control metrics using the metadata viewer module. Investigate the effect of clustering resolutions or compare annotation sources using cluster tree. Visualize gene expression using cell embeddings or in spatial context with feature plots and coexpression plots or in cell clusters using dotplots, violin plots. Manage marker tables from single-cell or pseudo-bulk analyses and track genes of interest using the gene scratchpad. Choose interesting cells with lasso selection directly from interactive plots and export for downstream analysis. Slice and dice datasets with custom filters based on cell metadata, gene expression or lasso selections. Efficiently handle large datasets using custom downsampling controls. CascadeR has cross-ecosystem compatibility with support for Seurat, SingleCellExperiment and AnnData objects. In addition, the app includes an integrated platform to manage multiple projects and user access that can be run on a central server to share with collaborators.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports anndata, ape, clustree, colorspace, data.table, dplyr, DT, ggplot2, ggraph, graphics, grid, htmltools, Matrix, methods, plotly, RColorBrewer, readr, reshape2, reticulate, rintrojs, rlang, scales, Seurat, SeuratObject, SingleCellExperiment, SummarizedExperiment, shiny, shinyBS, shinycssloaders, shinyjs, shinythemes, shinyWidgets, sortable, stats, tidyr, UpSetR, utils, viridis, yaml

Suggests BiocStyle, knitr, rmarkdown, shinymanager, testthat

VignetteBuilder knitr

BugReports <https://github.com/NICHD-BSPC/cascadeR/issues>

biocViews Software, SingleCell, Spatial, Transcriptomics,
GeneExpression, Visualization, DifferentialExpression,
ShinyApps, GUI

URL <https://nichd-bspc.github.io/cascadeR/>

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 libx11-dev zlib1g-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-09-18 22:52:46 UTC

RemoteUrl <https://github.com/BiocStaging/cascadeR>

RemoteRef HEAD

RemoteSha 3c944e1f7344b998eb09c0419f7bc952ec862347

Contents

add_cascade_analysis	3
BlendExpression2	5
BlendMap2	5
BuildClusterTree2	6
check_user_access	6
clustreemod	7
clustsummarymod	9
coexpmod	11
controlmod	14
create_access_yaml	15
dimredmod	16
dlfilemod	18
dlmod	19
dotplot	19
dotplotmod	20
feature_blend	22
feature_ly	24
ftplotmod	26
get_access_path	29
get_binned_exp	29
get_coexp_legend	30
get_coexp_tbl	31
get_coexpplt_colors	32
get_config	32
get_label_trace	33
get_limits	33
get_project_name_from_path	34

get_var_genes	35
helpmod	35
helpModal	36
install_cascade	37
linemod	37
make_example_seurat_object	38
markerpltmod	39
markertablegeneralmod	41
markertablemod	44
PseudoBulkExpression2	45
read_access_yaml	46
run_cascade	47
sanitize_colnames	47
save_access_yaml	48
scattermod	49
selmod	51
settingsmod	52
spatialcoexpmod	53
spatialftrpltmod	56
subsetServer	59
subsetUI	60
summarymod	61
umap_ly	62
violin_ly	63
violin2	64
violinmod	65

Index 68

add_cascade_analysis *Add new cascade analysis*

Description

Add new cascade analysis

Usage

```
add_cascade_analysis(
  obj_path,
  data_dir,
  project,
  analysis,
  cluster_markers = NULL,
  de_markers = NULL,
  conserved_markers = NULL,
  overwrite = FALSE,
  execute = FALSE
)
```

Arguments

<code>obj_path</code>	path to Seurat/anndata object
<code>data_dir</code>	output data directory. For convenience use the same data directory for all cascade projects.
<code>project</code>	project name. Creates a subfolder with this name inside <code>data_dir</code> if it doesn't exist.
<code>analysis</code>	analysis label. Creates a subfolder with this name inside <code>data_dir/project</code>
<code>cluster_markers</code>	(optional) path(s) to tab-delimited file(s) containing cluster markers. Output from Seurat's <code>FindAllMarkers</code> and scanpy's <code>rank_genes_groups</code> are supported.
<code>de_markers</code>	(optional) path(s) to tab-delimited file(s) containing differentially expressed markers from Seurat's <code>FindMarkers</code> or scanpy's <code>rank_genes_groups</code> .
<code>conserved_markers</code>	(optional) path(s) to tab-delimited file(s) containing conserved markers from Seurat's <code>FindConservedMarkers</code> function.
<code>overwrite</code>	boolean, if TRUE, existing analysis folder will be overwritten (default=FALSE)
<code>execute</code>	boolean, set this to TRUE to actually run the commands (default=FALSE)

Value

Invisibly returns NULL; called for the side effect of printing or executing setup commands.

Examples

```
local({
  data_dir <- tempfile("cascade-data-")
  dir.create(data_dir)

  obj_path <- tempfile("object-", fileext = ".rds")
  file.create(obj_path)

  on.exit(unlink(c(data_dir, obj_path), recursive = TRUE))

  add_cascade_analysis(
    obj_path = obj_path,
    data_dir = data_dir,
    project = "project1",
    analysis = "analysis1",
    execute = FALSE
  )
})
```

BlendExpression2	<i>Generalized version of Seurat:::BlendExpression</i>
------------------	--

Description

Seurat:::BlendExpression for arbitrary number of bins n = 1, means 10 bins n = 2, means 100 bins ...

Usage

BlendExpression2(data, n = 1)

Arguments

data	2-column matrix with gene expression to be blended
n	integer, defines number of bins = 10^n (default=1)

Details

NOTE: not used

Value

data.frame with blended expression

BlendMap2	<i>Interactive blend map plot</i>
-----------	-----------------------------------

Description

Seurat:::BlendMap function converted to plotly.

Usage

BlendMap2(color.matrix, dimnames = NULL)

Arguments

color.matrix	matrix of colors
dimnames	axis labels

Details

NOTE: this is not used

Value

plotly handle

BuildClusterTree2	<i>Build cluster tree for anndata objects</i>
-------------------	---

Description

This is a stripped down version of Seurat::BuildClusterTree for anndata objects. It takes the same arguments as the Seurat function.

Usage

```
BuildClusterTree2(
  object,
  features = NULL,
  dims = NULL,
  reduction = "X_pca",
  clust_column = "leiden"
)
```

Arguments

object	anndata object
features	genes to use for building cluster tree
dims	dimensions to use. This is only used if reduction is pca.
reduction	reduction to use for cluster tree
clust_column	column to use for grouping cells

Value

phylogenetic tree

check_user_access	<i>Get data areas a user has access to</i>
-------------------	--

Description

This function takes a username and returns a list with two elements:

Usage

```
check_user_access(al, u, admin = "admin")
```

Arguments

al	list with access settings; should have two elements - user_group & data_area
u	user name
admin	Admin user group

Details

user_group: one element vector data_area: vector of data areas

Value

list with user_group and data_area entries, or NULL if no access is found

Examples

```
# save access details to file
home <- Sys.getenv('HOME')

# create carnation data area if it doesn't exist
cascade_home <- file.path(home, 'cascade/data')
if(!dir.exists(cascade_home)) dir.create(cascade_home)

create_access_yaml(user = 'admin',
                  user_group = 'admin',
                  data_area = cascade_home)

# get current user access details
al <- read_access_yaml()

lst <- check_user_access(al, u='admin')
```

clustreemod

Cluster tree module

Description

Cluster tree module

Usage

```
clustreeUI(id, panel)
```

```
clustreeServer(id, obj, filtered, args, reload_global, config)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
obj	Cascade app object
filtered	cell barcodes for filtering object
args	reactive list with global args, 'grp_by' for grouping variable and 'dimred' for which dimension reduction to use
reload_global	reactive to reload module
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for the cluster tree Server called for the side effect of rendering cluster tree plots.

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = list(
    all = metadata_levels,
    filtered = metadata_levels
  ),
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  grp_by = "cluster",
  dimred = "umap"
)

config <- get_config()

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
```

```

    shiny::sidebarLayout(
      shiny::sidebarPanel(clustreeUI("clustree", "sidebar")),
      shiny::mainPanel(clustreeUI("clustree", "main"))
    )
  )
}

server <- function(input, output, session) {
  clustreeServer(
    "clustree",
    obj = app_object,
    filtered = shiny::reactive({ colnames(obj) }),
    args = shiny::reactive({ global_args }),
    reload_global = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )
}

shiny::shinyApp(ui, server)

```

clustsummarymod	<i>Metadata viewer module</i>
-----------------	-------------------------------

Description

Metadata viewer module

Usage

```
clustSummaryUI(id, panel)
```

```
clustSummaryServer(id, obj, filtered, args, reload_global, config)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
obj	Cascade app object
filtered	cell barcodes for filtering object
args	reactive list with elements, 'grp_by' for grouping variable and 'dimred' for which dimension reduction to use
reload_global	reactive to trigger reload
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for the metadata viewer Server called for the side effect of rendering metadata summary outputs.

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

numeric_cols <- vapply(metadata, is.numeric, logical(1))
metadata_numeric <- lapply(metadata[numeric_cols], function(x) {
  x <- x[!is.na(x)]
  hh <- graphics::hist(x, breaks = 20, plot = FALSE)
  data.frame(mids = hh$mids, counts = hh$counts)
})

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = list(
    all = metadata_levels,
    filtered = metadata_levels
  ),
  metadata_numeric = list(
    all = metadata_numeric,
    filtered = metadata_numeric
  ),
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  grp_by = "cluster",
  dimred = "umap"
)
```

```

)

config <- get_config()

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
  shiny::sidebarLayout(
    shiny::sidebarPanel(clustSummaryUI("metadata", "sidebar")),
    shiny::mainPanel(clustSummaryUI("metadata", "main"))
  )
)

server <- function(input, output, session) {
  clustSummaryServer(
    "metadata",
    obj = app_object,
    filtered = shiny::reactive({ colnames(obj) }),
    args = shiny::reactive({ global_args }),
    reload_global = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )
}

shiny::shinyApp(ui, server)

```

coexpmod

Coexpression plot module

Description

Coexpression plot module

Usage

```
coexpressionPlotUI(id, panel)
```

```

coexpressionPlotServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
  all_selected,
  show_selection,
  reset_selection,
  reload_global,
  refresh,

```

```
    config
  )
```

Arguments

<code>id</code>	Input id
<code>panel</code>	string, can be 'sidebar' or 'main'
<code>app_object</code>	Cascade app object
<code>filtered</code>	barcodes to filter object
<code>genes_to_plot</code>	reactive list with genes in scratchpad
<code>args</code>	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
<code>gene_choices</code>	reactive list with all genes present in object
<code>all_selected</code>	reactive containing list of selected points
<code>show_selection</code>	reactive to show selection
<code>reset_selection</code>	reactive to reset selection
<code>reload_global</code>	reactive to trigger reload
<code>refresh</code>	reactive to trigger plot refresh from sidebar button
<code>config</code>	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for coexpression plot Server returns reactive expression containing selected points from the coexpression plot

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
```

```

    stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
  })

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  assay = "RNA",
  slot = "data",
  grp_by = "cluster",
  dimred = "umap"
)

config <- get_config()

ui <- shiny::fluidPage(
  shiny::sidebarPanel(coexpressionPlotUI("coexpression", "sidebar")),
  shiny::mainPanel(
    shiny::tabsetPanel(coexpressionPlotUI("coexpression", "main"),
      shiny::verbatimTextOutput("selected")
    )
  )
)

server <- function(input, output, session) {
  selected <- coexpressionPlotServer(
    "coexpression",
    app_object = shiny::reactive({ app_object }),
    filtered = shiny::reactive({ colnames(obj) }),
    genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
    args = shiny::reactive({ global_args }),
    gene_choices = shiny::reactive({ rownames(obj) }),
    all_selected = shiny::reactive({ list() }),
    show_selection = shiny::reactive({ NULL }),
    reset_selection = shiny::reactive({ NULL }),
    reload_global = shiny::reactiveVal(0),
    refresh = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )

  output$selected <- shiny::renderPrint({
    selected()
  })
}

```

```
shiny::shinyApp(ui, server)
```

controlmod

Filter control module

Description

Filter control module

Usage

```
controlUI(id, label)
```

```
controlServer(id, full_obj, column, global, default = "all")
```

Arguments

id	string, input id
label	string, heading for controls
full_obj	reactive, object used to generate controls. can be data frame or list
column	string or reactive list, name of column or list element present in full_obj used to generate controls
global	reactive list used to set input externally
default	numeric vector or string, which elements to initialize with. If 'all' (default) all elements are selected, otherwise, can specify indices as numeric integer(s).

Value

UI returns filter controls UI Server returns reactive expression containing the selected filter values

Examples

```
metadata <- data.frame(
  cluster = factor(rep(c("A", "B"), each = 4), levels = c("A", "B")),
  condition = factor(rep(c("ctrl", "stim"), times = 4),
    levels = c("ctrl", "stim"))
)

ui <- shiny::fluidPage(
  controlUI("cluster_filter", "Cluster"),
  shiny::verbatimTextOutput("selected")
)

server <- function(input, output, session) {
  selected <- controlServer(
    "cluster_filter",
    full_obj = shiny::reactive({ metadata })
  )
}
```

```

      column = "cluster",
      global = shiny::reactive({ NULL })
    )

    output$selected <- shiny::renderPrint({
      selected()
    })
  }

  shiny::shinyApp(ui, server)

```

create_access_yaml	<i>Create access yaml</i>
--------------------	---------------------------

Description

This function creates an access yaml file. This is primarily intended for the first run.

Usage

```
create_access_yaml(user, user_group, data_area)
```

Arguments

user	User name
user_group	User group
data_area	Path to data area containing RDS files

Value

Invisibly returns NULL; called for the side effect of writing the access yaml file.

Examples

```

local({
  access_dir <- tempfile("cascade-access-")
  dir.create(access_dir)

  old <- Sys.getenv("CASCADE_ACCESS_YAML", unset = NA_character_)
  Sys.setenv(CASCADE_ACCESS_YAML = access_dir)
  on.exit({
    if (is.na(old)) {
      Sys.unsetenv("CASCADE_ACCESS_YAML")
    } else {
      Sys.setenv(CASCADE_ACCESS_YAML = old)
    }
    unlink(access_dir, recursive = TRUE)
  })
})

```

```

    create_access_yaml("user1", "lab1", "/data/lab1")
  })

```

dimredmod

Cell embeddings module

Description

Cell embeddings module

Usage

```
dimredUI(id, panel)
```

```

dimredServer(
  id,
  obj,
  filtered,
  args,
  all_selected,
  show_selection,
  reset_selection,
  reload_global,
  config
)

```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
obj	Cascade app object
filtered	barcodes to filter object
args	reactive list with global args, 'grp_by' for grouping variable and 'dimred' for which dimension reduction to use
all_selected	reactive containing list of selected points
show_selection	reactive to show selection
reset_selection	reactive to reset selection
reload_global	reactive to trigger reload
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for cell embeddings Server returns reactive expression containing selected points from embedding plots

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = list(
    all = metadata_levels,
    filtered = metadata_levels
  ),
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  grp_by = "cluster",
  dimred = "umap"
)

config <- get_config()

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
```

```

shiny::sidebarLayout(
  shiny::sidebarPanel(dimredUI("dimred", "sidebar")),
  shiny::mainPanel(
    dimredUI("dimred", "main"),
    shiny::verbatimTextOutput("selected")
  )
)

server <- function(input, output, session) {
  selected <- dimredServer(
    "dimred",
    obj = app_object,
    filtered = shiny::reactive({ colnames(obj) }),
    args = shiny::reactive({ global_args }),
    all_selected = shiny::reactive({ list() }),
    show_selection = shiny::reactive({ NULL }),
    reset_selection = shiny::reactive({ NULL }),
    reload_global = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )

  output$selected <- shiny::renderPrint({
    selected()
  })
}

shiny::shinyApp(ui, server)

```

dlfilemod

*Download file module***Description**

UI & server for download file module

Usage

```
downloadFileUI(id)
```

```
downloadFileServer(id, data, file_type)
```

Arguments

id	Input id
data	reactive w data frame
file_type	reactive/static value used for output filename

Value

Shiny UI & server for the file download button module

dlmod	<i>Download button module</i>
-------	-------------------------------

Description

Download button module

Usage

```
downloadPlotUI(id)
```

```
downloadPlotServer(id, outplot, plot_type, direction = NULL)
```

Arguments

id	Input id
outplot	reactive plot handle
plot_type	reactive/static value used for output filename
direction	direction of phylogenetic tree

Value

Shiny UI & server element for the plot download button module

dotplot	<i>Dotplot</i>
---------	----------------

Description

Lightweight dotplot reimplementation to run only on data frame with gene expression and metadata.

Usage

```
dotplot(  
  df,  
  xcol,  
  ycol,  
  split = NULL,  
  scale = TRUE,  
  dot.scale = 10,  
  col.min = -2.5,  
  col.max = 2.5  
)
```

Arguments

<code>df</code>	data.frame with gene expression data
<code>xcol</code>	x-axis grouping variable
<code>ycol</code>	genes to show (on y-axis). can be multiple
<code>split</code>	faceting variable
<code>scale</code>	should data be scaled? Default: TRUE
<code>dot.scale</code>	dot size scaling factor (default: 10)
<code>col.min</code>	if data is scaled, this is the lower limit of values (default: -2.5)
<code>col.max</code>	if data is scaled, this is the upper limit of values (default: 2.5)

Value

ggplot2 handle

Examples

```
df <- data.frame(
  cluster = factor(rep(c("A", "B"), each = 4)),
  condition = factor(rep(c("ctrl", "stim"), times = 4)),
  GeneA = c(0, 2, 4, 5, 1, 3, 6, 8),
  GeneB = c(1, 0, 3, 2, 5, 0, 7, 4)
)

dotplot(df, xcol = "cluster", ycol = c("GeneA", "GeneB"), scale = FALSE)
dotplot(df, xcol = "cluster", ycol = "GeneA", split = "condition")
```

dotplotmod

Dotplot module

Description

Dotplot module

Usage

```
dotPlotUI(id, panel)

dotPlotServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
```

```

    reload_global,
    refresh,
    config
  )

```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for dotplot module Server called for the side effect of rendering a dotplot.

Examples

```

# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

```

```

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  assay = "RNA",
  slot = "data",
  grp_by = "cluster",
  dimred = "umap"
)

config <- get_config()

ui <- shiny::fluidPage(
  shiny::sidebarPanel(dotPlotUI("dotplot", "sidebar")),
  shiny::mainPanel(shiny::tabsetPanel(dotPlotUI("dotplot", "main")))
)

server <- function(input, output, session) {
  dotPlotServer(
    "dotplot",
    app_object = shiny::reactive({ app_object }),
    filtered = shiny::reactive({ colnames(obj) }),
    genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
    args = shiny::reactive({ global_args }),
    gene_choices = shiny::reactive({ rownames(obj) }),
    reload_global = shiny::reactiveVal(0),
    refresh = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )
}

shiny::shinyApp(ui, server)

```

feature_blend

Binned coexpression plot for 2 features

Description

This is a wrapper around `umap_ly` that uses four colors for four categories instead of a blend matrix.

Usage

```

feature_blend(
  df,
  xcol,
  ycol,
  blend_cols,
  colors,
  split = NULL,
  col_threshold_1 = 0.5,
  col_threshold_2 = 0.5,
  n = 100,
  bin_mode = "range",
  neutral_color = "lightgray",
  showlegend = TRUE,
  showticklabels = TRUE,
  type = "scattergl",
  marker_size = 2,
  margin = 0.03,
  alpha = 0.3,
  free_axes = FALSE,
  width = NULL,
  height = NULL,
  source = "A"
)

```

Arguments

df	data.frame with plotting data
xcol	x-axis coordinates
ycol	y-axis coordinates
blend_cols	columns to blend
colors	3 color vector (geneA, geneB, merge)
split	metadata column to split plot by
col_threshold_1	blending threshold for gene 1
col_threshold_2	blending threshold for gene 2
n	number of bins for color range
bin_mode	how to bin? can be 'range' (gene expression binned into n bins) or 'quantile' (cells binned into n bins by exp)
neutral_color	double negative color
showlegend	should legend be shown?
showticklabels	should tick labels be drawn?
type	type of plotly trace (default: 'scattergl')

marker_size	size of markers
margin	margin of plot (should be between 0 & 1)
alpha	marker opacity
free_axes	should subplots have free axes?
width	width of plot in pixels. If NULL (default), plot is auto-sized.
height	height of plot in pixels. if NULL (default), plot is auto-sized.
source	name of source to return data from

Value

list with plot data and plotly handle

Examples

```
df <- data.frame(
  UMAP_1 = c(-1, 0, 1, 2),
  UMAP_2 = c(0, 1, 0, -1),
  GeneA = c(0, 5, 0, 10),
  GeneB = c(0, 0, 8, 10)
)

blended <- feature_blend(
  df,
  xcol = "UMAP_1",
  ycol = "UMAP_2",
  blend_cols = c("GeneA", "GeneB"),
  colors = c("#ff0000", "#0000ff", "#ff00ff"),
  n = 10
)
names(blended)
blended$data
```

feature_ly	<i>Interactive feature plot</i>
------------	---------------------------------

Description

Interactive feature plot

Usage

```
feature_ly(
  df,
  xcol,
  ycol,
  color,
```

```

    colors,
    crange = NULL,
    split = NULL,
    label_cols = NULL,
    row_view = "auto",
    showscale = TRUE,
    reversescale = FALSE,
    showticklabels = TRUE,
    marker_size = 2,
    margin = 0.03,
    alpha = 0.3,
    free_axes = FALSE,
    reorder = TRUE,
    width = NULL,
    height = NULL,
    title_mode = "color",
    source = "A"
)

```

Arguments

<code>df</code>	data.frame with expression data & plot coordinates
<code>xcol</code>	x-axis coordinates
<code>ycol</code>	y-axis coordinates
<code>color</code>	color column. should be numeric
<code>colors</code>	plotly color scale. should be continuous
<code>crange</code>	range of gene expression to show on plot
<code>split</code>	column to split plots by
<code>label_cols</code>	Additional metadata columns to use for labeling pts
<code>row_view</code>	how should plots be laid out? can be 'single' or 'auto'
<code>showscale</code>	show the color scale?
<code>reversescale</code>	should we reverse the color scale?
<code>showticklabels</code>	should tick labels be drawn?
<code>marker_size</code>	marker size
<code>margin</code>	plot margin
<code>alpha</code>	marker opacity
<code>free_axes</code>	should subplots have free axes?
<code>reorder</code>	should be sort cells in ascending order of expression?
<code>width</code>	width of plot in pixels
<code>height</code>	height of plot in pixels
<code>title_mode</code>	string specifying how axes should be titled. If 'color' (default), the y-axis is titled with the coloring variable; if 'xy', xcol and ycol are used as labels.
<code>source</code>	name of source to return data from

Value

plotly handle

Examples

```
df <- data.frame(  
  UMAP_1 = c(-1, 0, 1, 2),  
  UMAP_2 = c(0, 1, 0, -1),  
  GeneA = c(0, 2, 4, 8)  
)  
  
feature_ly(  
  df,  
  xcol = "UMAP_1",  
  ycol = "UMAP_2",  
  color = "GeneA",  
  colors = "Viridis",  
  crange = c(0, 8)  
)
```

ftrpltmod

Feature plot module

Description

Feature plot module

Usage

```
featurePlotUI(id, panel)  
  
featurePlotServer(  
  id,  
  app_object,  
  filtered,  
  genes_to_plot,  
  args,  
  gene_choices,  
  all_selected,  
  show_selection,  
  reset_selection,  
  reload_global,  
  refresh,  
  config  
)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
all_selected	reactive containing list of selected points
show_selection	reactive to show selection
reset_selection	reactive to reset selection
reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for feature plot Server returns reactive expression containing selected points from the feature plot

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
```

```

    rds = obj,
    obj_type = "seurat",
    metadata = metadata,
    metadata_levels = metadata_levels,
    cluster_colors = cluster_colors,
    grouping_vars = grouping_vars,
    spatial_coords = NULL,
    imagerow_max = NULL,
    imagerow_min = NULL
  )

  global_args <- list(
    assay = "RNA",
    slot = "data",
    grp_by = "cluster",
    dimred = "umap"
  )

  config <- get_config()

  ui <- shiny::fluidPage(
    shinyjs::useShinyjs(),
    shiny::sidebarLayout(
      shiny::sidebarPanel(featurePlotUI("feature", "sidebar")),
      shiny::mainPanel(
        shiny::tabsetPanel(featurePlotUI("feature", "main"),
          shiny::verbatimTextOutput("selected")
        )
      )
    )
  )

  server <- function(input, output, session) {
    selected <- featurePlotServer(
      "feature",
      app_object = shiny::reactive({ app_object }),
      filtered = shiny::reactive({ colnames(obj) }),
      genes_to_plot = shiny::reactive({ "GeneA" }),
      args = shiny::reactive({ global_args }),
      gene_choices = shiny::reactive({ rownames(obj) }),
      all_selected = shiny::reactive({ list() }),
      show_selection = shiny::reactive({ NULL }),
      reset_selection = shiny::reactive({ NULL }),
      reload_global = shiny::reactiveVal(0),
      refresh = shiny::reactiveVal(0),
      config = shiny::reactive({ config })
    )

    output$selected <- shiny::renderPrint({
      selected()
    })
  }

  shiny::shinyApp(ui, server)

```

get_access_path	<i>Get path to access yaml</i>
-----------------	--------------------------------

Description

This function checks for an environment variable 'CASCADE_ACCESS_YAML' to specify directory containing access yaml. If env variable does not exist uses home directory as save location.

Usage

```
get_access_path()
```

Value

path to access yaml

get_binned_exp	<i>Bin gene expression in columns of a data frame</i>
----------------	---

Description

The expression of a gene present in columns of a data frame are binned based either on the range of observed expression ('range') or quantiles of cells where the gene is expressed.

Usage

```
get_binned_exp(df, bins = 100, mode = "range")
```

Arguments

df	data.frame with expression data
bins	number of bins
mode	how to calculate bins? options are 'range' or 'quantile'

Value

data.frame

`get_coexp_legend`*Function to generate coexpression plot legend*

Description

This interactive legend is shown in the sidebar to help visually select coexpression blend threshold.

Usage

```
get_coexp_legend(  
  colors,  
  dimnames,  
  xline,  
  yline,  
  n = 100,  
  neutral_color = "lightgray",  
  margin = 0.03  
)
```

Arguments

<code>colors</code>	3-color vector (geneA, geneB, both)
<code>dimnames</code>	names to show on x & y axis
<code>xline</code>	x-axis threshold
<code>yline</code>	y-axis threshold
<code>n</code>	number of bins
<code>neutral_color</code>	color to use for negative cells
<code>margin</code>	plot margin

Value

plotly handle

Examples

```
get_coexp_legend(  
  colors = c("#ff0000", "#0000ff", "#ff00ff"),  
  dimnames = c("GeneA", "GeneB"),  
  xline = 0.25,  
  yline = 0.75,  
  n = 20,  
  neutral_color = "#eeeeee"  
)
```

get_coexp_tbl	<i>Get counts table based on coexpression</i>
---------------	---

Description

Get data frame with counts of cells binned into four categories based on expression of two genes

Usage

```
get_coexp_tbl(  
  df,  
  genes,  
  n = 100,  
  threshold1 = 0.5,  
  threshold2 = 0.5,  
  bin_mode = "range"  
)
```

Arguments

df	data.frame with gene expression data
genes	genes to calculate coexpression for
n	number of bins
threshold1	percentile to define expression for gene 1
threshold2	percentile to define expression for gene 2
bin_mode	how to bin? can be 'range' (gene expression binned into n bins) or 'quantile' (cells binned into n bins by expression quantiles)

Details

- both genes low
- gene A low, gene B high
- gene A high, gene B low
- gene A high, gene B high

Value

data.frame with co-expression counts

Examples

```
df <- data.frame(
  GeneA = c(0, 10, 0, 10),
  GeneB = c(0, 0, 10, 10)
)

get_coexp_tbl(df, genes = c("GeneA", "GeneB"), n = 10)
```

get_coexplt_colors	<i>Function to return coexpression plot colormaps</i>
--------------------	---

Description

Function to return coexpression plot colormaps

Usage

```
get_coexplt_colors(x)
```

Arguments

`x` character, name of color map. Can be 'red-blue' or 'red-green'

Value

4 color vector: (neutral, gene A, gene B, merge)

get_config	<i>Get config</i>
------------	-------------------

Description

This function reads the config.yaml and returns the list

Usage

```
get_config()
```

Value

list containing config items

Examples

```
cfg <- get_config()
```

get_label_trace	<i>Get plotly label trace</i>
-----------------	-------------------------------

Description

Plotly trace to add labels to existing plot.

Usage

```
get_label_trace(plot_data, labeled_pts, split = FALSE)
```

Arguments

plot_data	plotly plot data
labeled_pts	points to label
split	column by which plot is split

Value

plotly handle

get_limits	<i>Get limits of numeric vector</i>
------------	-------------------------------------

Description

This function calculates the limits of a numeric vector and returns them after rounding to specified number of significant digits.

Usage

```
get_limits(gq, sig_digits = 3, na.rm = TRUE)
```

Arguments

gq	numeric vector
sig_digits	significant digits for rounding
na.rm	remove NAs? Default: TRUE

Value

vector of length 2 with min, max

```
get_project_name_from_path
```

Get project name from path

Description

This function takes in a path to an RDS file and returns a string to be used as project name

Usage

```
get_project_name_from_path(  
  x,  
  depth = 2,  
  end_offset = 0,  
  staging_dir = "staged",  
  fsep = .Platform$file.sep  
)
```

Arguments

x	character path to RDS file
depth	integer how many levels below path to look?
end_offset	integer how far from the end of path to end?
staging_dir	name of staging directory
fsep	file separator to split path with

Details

So if the path is: /path/to/project/test/clustered.Rds and depth=2 & end_offset=0 this returns: project/test

Value

project name

Examples

```
get_project_name_from_path(  
  "/path/to/project/test/clustered.Rds",  
  depth = 2,  
  fsep = "/"  
)  
  
get_project_name_from_path(  
  "/path/project/staged/test/clustered.Rds",  
  depth = 2,  
  staging_dir = "staged",  
  fsep = "/"
```

)

get_var_genes	<i>Get highly variable genes</i>
---------------	----------------------------------

Description

This function calculates most variable genes from either Seurat or anndata objects.

Usage

```
get_var_genes(obj, obj_type, assay_name)
```

Arguments

obj	object to get variable genes from
obj_type	type of object being used. can be 'seurat' or 'anndata'
assay_name	name of assay for variable genes. Ignored if obj_type == 'anndata'.

Value

vector of gene names

helpmod	<i>Help button module</i>
---------	---------------------------

Description

Module UI & server for help buttons.

Usage

```
helpButtonUI(id)
helpButtonServer(id, ...)
```

Arguments

id	Module id. This also doubles as prefixes for help text files.
...	other params passed to helpModal()

Value

UI returns tagList with help button UI. Server invisibly returns NULL (used for side effects).

Examples

```
library(shiny)

# app with a single help button to show DE summary table details
if(interactive()){
  shinyApp(
    ui = fluidPage(
      helpButtonUI('dimred_help')
    ),
    server = function(input, output, session){
      helpButtonServer('dimred_help')
    }
  )
}
```

helpModal	<i>Help modal</i>
-----------	-------------------

Description

This generates a modal dialog that includes text from a markdown file.

Usage

```
helpModal(mdfile, title = NULL, ...)
```

Arguments

mdfile	path to markdown file
title	Title of modal dialog
...	other params passed to modalDialog()

Value

Shiny modal dialog

install_cascade	<i>Create cascade python environment</i>
-----------------	--

Description

This function installs 'plotly' and 'kaleido' python packages in an environment to allow PDF downloads from plotly plots, and 'anndata' to allow h5ad files.

Usage

```
install_cascade(envname, ...)
```

Arguments

envname	name of the python environment
...	parameters passed to reticulate::py_install

Value

Return value from reticulate::py_install(); called for the side effect of installing Python packages.

Examples

```
install_cascade()
```

linemod	<i>Line plot module</i>
---------	-------------------------

Description

Line plot module

Usage

```
linePlotUI(id, panel)
```

```
linePlotServer(  
  id,  
  app_object,  
  filtered,  
  genes_to_plot,  
  args,  
  gene_choices,  
  reload_global,
```

```

    refresh,
    config
  )

```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

Shiny UI & server elements for the line plot module

Shiny module server return value; called for the side effect of rendering a line plot.

```
make_example_seurat_object
```

Create a tiny example Seurat object

Description

This creates a small normalized Seurat object with an RNA assay, metadata, identities, and a two-dimensional UMAP reduction. It is intended for documentation examples and lightweight module demos.

Usage

```

make_example_seurat_object(
  genes = paste0("Gene", LETTERS[seq_len(4)]),
  cells = paste0("cell", seq_len(8)),
  assay = "RNA"
)

```

Arguments

genes	character vector of gene names to include.
cells	character vector of cell names to include.
assay	assay name to use when creating the object.

Value

A Seurat object.

Examples

```
obj <- make_example_seurat_object()
obj
```

markerplmod	<i>Marker plots module</i>
-------------	----------------------------

Description

Marker plots module

Usage

```
markerPlotUI(id, panel)
```

```
markerPlotServer(  
  id,  
  obj,  
  filtered,  
  genes_to_plot,  
  args,  
  gene_choices,  
  all_selected,  
  show_selection,  
  reset_selection,  
  reload_global,  
  config  
)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
obj	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
all_selected	reactive containing list of selected points

```

show_selection  reactive to show selection
reset_selection
                reactive to reset selection

reload_global  reactive to trigger reload
config         reactive list with config settings

```

Value

UI returns sidebar/main panel UI elements for marker plots Server returns reactive expression containing selected points from marker plot modules

Examples

```

# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = list(
    all = metadata_levels,
    filtered = metadata_levels
  ),
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  assay = "RNA",

```

```

    slot = "data",
    grp_by = "cluster",
    dimred = "umap"
  )

  config <- get_config()

  ui <- shiny::fluidPage(
    shinyjs::useShinyjs(),
    shiny::sidebarLayout(
      shiny::sidebarPanel(markerPlotUI("markers", "sidebar")),
      shiny::mainPanel(
        markerPlotUI("markers", "main"),
        shiny::verbatimTextOutput("selected")
      )
    )
  )

  server <- function(input, output, session) {
    selected <- markerPlotServer(
      "markers",
      obj = app_object,
      filtered = shiny::reactive({ colnames(obj) }),
      genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
      args = shiny::reactive({ global_args }),
      gene_choices = shiny::reactive({ rownames(obj) }),
      all_selected = shiny::reactive({ list() }),
      show_selection = shiny::reactive({ NULL }),
      reset_selection = shiny::reactive({ NULL }),
      reload_global = shiny::reactiveVal(0),
      config = shiny::reactive({ config })
    )

    output$selected <- shiny::renderPrint({
      selected()
    })
  }

  shiny::shinyApp(ui, server)

```

markertablegeneralmod *Generalized marker table module*

Description

Generalized marker table module

Usage

```
markerTableGeneralUI(id, panel, type, label = NULL)

markerTableGeneralServer(
  id,
  obj,
  type,
  genes_to_plot,
  reset_genes,
  global_args,
  args,
  reload_global,
  config
)
```

Arguments

id	Input id
panel	string, can be 'sidebar', 'selection' or 'main'
type	string, used to define type of table
label	string, label for table
obj	reactive list with marker tables
genes_to_plot	reactive list of genes to plot
reset_genes	reactive to trigger gene selection reset
global_args	reactive list with global settings
args	reactive list with 'max_padj', 'max_lfc'
reload_global	reactive to trigger global args reload
config	reactive list with config settings

Value

UI returns sidebar/selection/main panel UI elements for a marker table Server returns reactive expression containing marker filters and selected genes

Examples

```
marker_tbl <- data.frame(
  gene = c("GeneA", "GeneB", "GeneC", "GeneD"),
  cluster = c("A", "A", "B", "B"),
  avg_log2FC = c(1.2, 0.7, 1.1, 0.8),
  pct.1 = c(0.9, 0.8, 0.85, 0.75),
  pct.2 = c(0.2, 0.3, 0.25, 0.35),
  p_val_adj = c(0.001, 0.02, 0.005, 0.03)
)

config <- get_config()
```

```

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
  shiny::sidebarLayout(
    shiny::sidebarPanel(
      markerTableGeneralUI(
        "allmarkers",
        panel = "sidebar",
        type = "allmarkers",
        label = "Cluster Markers"
      ),
      markerTableGeneralUI(
        "allmarkers",
        panel = "selection",
        type = "allmarkers",
        label = "Cluster Markers"
      )
    ),
    shiny::mainPanel(
      shiny::tabsetPanel(
        markerTableGeneralUI(
          "allmarkers",
          panel = "main",
          type = "allmarkers",
          label = "Cluster Markers"
        )
      ),
      shiny::verbatimTextOutput("selected")
    )
  )
)

server <- function(input, output, session) {
  selected <- markerTableGeneralServer(
    "allmarkers",
    obj = shiny::reactive({ list(markers = marker_tbl) }),
    type = "allmarkers",
    genes_to_plot = shiny::reactive({ character() }),
    reset_genes = shiny::reactive({ NULL }),
    global_args = shiny::reactive({ list() }),
    args = shiny::reactive({ list(max_padj = 0.1, min_lfc = 0) }),
    reload_global = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )

  output$selected <- shiny::renderPrint({
    selected()
  })
}

shiny::shinyApp(ui, server)

```

markertablemod	<i>Marker tables module</i>
----------------	-----------------------------

Description

This is a master wrapper for generalized marker tables.

Usage

```
markerTableUI(id, panel)
```

```
markerTableServer(id, obj, genes_to_plot, reset_genes, reload_global, config)
```

Arguments

id	Input id
panel	string, can be 'global', 'sidebar', 'selection' or 'main'
obj	Cascade app object
genes_to_plot	reactive with list of genes to be plotted/selected
reset_genes	reactive to trigger gene selection reset
reload_global	reactive to trigger reload
config	reactive list with config settings

Value

UI returns global/sidebar/selection/main panel UI elements for marker tables Server returns reactive expression containing selected marker genes

Examples

```
# example obj
obj <- make_example_seurat_object()

marker_tbl <- data.frame(
  gene = c("GeneA", "GeneB", "GeneC", "GeneD"),
  cluster = c("A", "A", "B", "B"),
  avg_log2FC = c(1.2, 0.7, 1.1, 0.8),
  pct.1 = c(0.9, 0.8, 0.85, 0.75),
  pct.2 = c(0.2, 0.3, 0.25, 0.35),
  p_val_adj = c(0.001, 0.02, 0.005, 0.03)
)

app_object <- list(
  rds = obj,
  allmarkers = marker_tbl,
  consmarkers = NULL,
  demarkers = NULL
)
```

```

)

config <- get_config()

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
  shiny::sidebarLayout(
    shiny::sidebarPanel(
      markerTableUI("markers", "global"),
      markerTableUI("markers", "sidebar"),
      markerTableUI("markers", "selection")
    ),
    shiny::mainPanel(
      markerTableUI("markers", "main"),
      shiny::verbatimTextOutput("selected")
    )
  )
)

server <- function(input, output, session) {
  selected <- markerTableServer(
    "markers",
    obj = app_object,
    genes_to_plot = shiny::reactive({ character() }),
    reset_genes = shiny::reactive({ NULL }),
    reload_global = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )

  output$selected <- shiny::renderPrint({
    selected()
  })
}

shiny::shinyApp(ui, server)

```

PseudoBulkExpression2 *Get pseudobulk expression from anndata object*

Description

This is a stripped-down lightweight version of Seurat's internal function adapted for an anndata object that has genes in columns not rows.

Usage

```

PseudoBulkExpression2(
  object,
  pb.method = "average",

```

```
    features = NULL,  
    group.by = "ident"  
  )
```

Arguments

object	anndata object
pb.method	pseudo-bulk method. Can be 'average' (default) or 'aggregate'.
features	features to analyze. Default is all features in the object
group.by	metadata column to group cells by

Value

matrix with pseudo-bulk counts

read_access_yaml	<i>Read access yaml with user groups and data areas</i>
------------------	---

Description

This function reads the access yaml file and returns user groups and data areas as a list of data frames.

Usage

```
read_access_yaml()
```

Value

list with user group and data area settings

Examples

```
local({  
  access_dir <- tempfile("cascade-access-")  
  dir.create(access_dir)  
  
  old <- Sys.getenv("CASCADE_ACCESS_YAML", unset = NA_character_)  
  Sys.setenv(CASCADE_ACCESS_YAML = access_dir)  
  on.exit({  
    if (is.na(old)) {  
      Sys.unsetenv("CASCADE_ACCESS_YAML")  
    } else {  
      Sys.setenv(CASCADE_ACCESS_YAML = old)  
    }  
    unlink(access_dir, recursive = TRUE)  
  })  
})
```

```
  create_access_yaml("user1", "lab1", "/data/lab1")
  read_access_yaml()
})
```

run_cascade

Cascade

Description

Interactive shiny dashboard for exploring single-cell or spatial RNA-Seq data.

Usage

```
run_cascade(credentials = NULL, passphrase = NULL, enable_admin = TRUE, ...)
```

Arguments

credentials	path to encrypted sqlite db with user credentials.
passphrase	passphrase for credentials db.
enable_admin	if TRUE, admin view is shown. Note, this is only available if credentials have sqlite backend.
...	parameters passed to shinyApp() call

Value

Shiny app object

Examples

```
shiny::runApp(
  run_cascade()
)
```

sanitize_colnames

Sanitize strings

Description

This function removes special characters from strings. Meant to be used to sanitize column names of a data frame before plotting.

Usage

```
sanitize_colnames(cnames, bad_char = "\\W+", repl = "_")
```

Arguments

cnames	column names to sanitize
bad_char	regex with characters to be replaced with 'repl'
repl	replacement character

Value

sanitized string vector

save_access_yaml	<i>Save access yaml to file</i>
------------------	---------------------------------

Description

This function saves access details (user groups and data areas) to the designated access yaml file.

Usage

save_access_yaml(lst)

Arguments

lst	list of data frames with user_groups and data_areas
-----	---

Value

Invisibly returns NULL; called for the side effect of writing the access yaml file.

Examples

```
local({
  access_dir <- tempfile("cascade-access-")
  dir.create(access_dir)

  old <- Sys.getenv("CASCADE_ACCESS_YAML", unset = NA_character_)
  Sys.setenv(CASCADE_ACCESS_YAML = access_dir)
  on.exit({
    if (is.na(old)) {
      Sys.unsetenv("CASCADE_ACCESS_YAML")
    } else {
      Sys.setenv(CASCADE_ACCESS_YAML = old)
    }
    unlink(access_dir, recursive = TRUE)
  })

  save_access_yaml(list(
    user_group = list(user1 = "lab1", user2 = "lab2"),
    data_area = list(lab1 = "/data/lab1", lab2 = "/data/lab2")
  ))
})
```

```

    ))
    read_access_yaml()
  })

```

scattermod

Gene-gene scatter plot module

Description

Gene-gene scatter plot module

Usage

```
scatterPlotUI(id, panel)
```

```

scatterPlotServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
  all_selected,
  show_selection,
  reset_selection,
  reload_global,
  refresh,
  config
)

```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
all_selected	reactive containing list of selected points
show_selection	reactive to show selection
reset_selection	reactive to reset selection

reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for scatter plot Server returns reactive expression containing selected points from the scatter plot

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = NULL,
  imagerow_max = NULL,
  imagerow_min = NULL
)

global_args <- list(
  assay = "RNA",
  slot = "data",
  grp_by = "cluster",
  dimred = "umap",
  assay_list = list(RNA = c("counts", "data"))
)
```

```

config <- get_config()

ui <- shiny::fluidPage(
  shinyjs::useShinyjs(),
  shiny::sidebarLayout(
    shiny::sidebarPanel(scatterPlotUI("scatter", "sidebar")),
    shiny::mainPanel(
      shiny::tabsetPanel(scatterPlotUI("scatter", "main")),
      shiny::verbatimTextOutput("selected")
    )
  )
)

server <- function(input, output, session) {
  selected <- scatterPlotServer(
    "scatter",
    app_object = shiny::reactive({ app_object }),
    filtered = shiny::reactive({ colnames(obj) }),
    genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
    args = shiny::reactive({ global_args }),
    gene_choices = shiny::reactive({ rownames(obj) }),
    all_selected = shiny::reactive({ list() }),
    show_selection = shiny::reactive({ NULL }),
    reset_selection = shiny::reactive({ NULL }),
    reload_global = shiny::reactiveVal(0),
    refresh = shiny::reactiveVal(0),
    config = shiny::reactive({ config })
  )

  output$selected <- shiny::renderPrint({
    selected()
  })
}

shiny::shinyApp(ui, server)

```

selmod

Download selection module

Description

UI & server for selection module

Usage

```
selectionUI(id)
```

```
selectionServer(id, selections, obj)
```

Arguments

id	Input id
selections	reactive list w selections
obj	reactive app object

Value

Shiny UI & server for the download selection button module

settingsmod	<i>Settings module</i>
-------------	------------------------

Description

Module UI & server for user access details interface.

Usage

```
settingsUI(id, panel, username)

settingsServer(id, details, depth, end_offset, assay_fun, config)
```

Arguments

id	Module id
panel	context for generating ui elements ('sidebar' or 'main')
username	user name
details	reactive list with user name & app location details
depth	project name depth
end_offset	project name end offset
assay_fun	function to parse assay names from file path
config	reactive list with config settings

Value

UI returns tagList with module UI Server returns reactive with list containing user access details

Examples

```

library(shiny)

# default username
username <- reactive({ NULL })

# internal carnation config
config <- reactiveVal(get_config())

# regex to find carnation files
pattern <- reactive({ config()$server$pattern })

# access permissions
assay.list <- reactiveValues(l=read_access_yaml())

if(interactive()){
  shinyApp(
    ui = fluidPage(
      sidebarPanel(uiOutput('settings_sidebar')),
      mainPanel(uiOutput('settings_main'))
    ),
    server = function(input, output, session){
      output$settings_main <- renderUI({
        settingsUI('settings', panel='main', username=username)
      })

      output$settings_sidebar <- renderUI({
        settingsUI('settings', panel='sidebar', username=username)
      })

      settings <- settingsServer('p',
                                details=reactive({
                                  list(username=username,
                                       where=NULL)
                                }),
                                depth=3,
                                end_offset=1,
                                assay_fun=function(x) basename(dirname(x)),
                                config
                                )
    }
  )
}

```

Description

Spatial coexpression plot module

Usage

```
spatialCoexpressionPlotUI(id, panel)
```

```
spatialCoexpressionPlotServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
  slice,
  all_selected,
  show_selection,
  reset_selection,
  reload_global,
  refresh,
  config
)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
slice	reactive with slices to be used for plotting
all_selected	reactive containing list of selected points
show_selection	reactive to show selection
reset_selection	reactive to reset selection
reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for spatial coexpression plot Server returns reactive expression containing selected points from the spatial coexpression plot

Examples

```

# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

spatial_coords <- data.frame(
  rn = colnames(obj),
  slice = as.character(metadata$orig.ident),
  imagecol = rep(c(1, 2, 1, 2), length.out = ncol(obj)),
  imagerow = rep(c(1, 1, 2, 2), length.out = ncol(obj)),
  stringsAsFactors = FALSE
)
imagerow_max <- as.list(
  tapply(spatial_coords$imagerow, spatial_coords$slice, max)
)
imagerow_min <- as.list(
  tapply(spatial_coords$imagerow, spatial_coords$slice, min)
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = spatial_coords,
  imagerow_max = imagerow_max,
  imagerow_min = imagerow_min
)

global_args <- list(
  assay = "RNA",
  slot = "data",

```

```

    grp_by = "cluster",
    dimred = "umap"
  )

  config <- get_config()

  ui <- shiny::fluidPage(
    shinyjs::useShinyjs(),
    shiny::sidebarLayout(
      shiny::sidebarPanel(spatialCoexpressionPlotUI("spatial_coexp", "sidebar")),
      shiny::mainPanel(
        shiny::tabsetPanel(spatialCoexpressionPlotUI("spatial_coexp", "main")),
        shiny::verbatimTextOutput("selected")
      )
    )
  )

  server <- function(input, output, session) {
    selected <- spatialCoexpressionPlotServer(
      "spatial_coexp",
      app_object = shiny::reactive({ app_object }),
      filtered = shiny::reactive({ colnames(obj) }),
      genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
      args = shiny::reactive({ global_args }),
      gene_choices = shiny::reactive({ rownames(obj) }),
      slice = shiny::reactive({ unique(spatial_coords$slice)[1] }),
      all_selected = shiny::reactive({ list() }),
      show_selection = shiny::reactive({ NULL }),
      reset_selection = shiny::reactive({ NULL }),
      reload_global = shiny::reactiveVal(0),
      refresh = shiny::reactiveVal(0),
      config = shiny::reactive({ config })
    )

    output$selected <- shiny::renderPrint({
      selected()
    })
  }

  shiny::shinyApp(ui, server)

```

Description

Spatial feature plot module

Usage

```
spatialFeaturePlotUI(id, panel)
```

```
spatialFeaturePlotServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
  slice,
  all_selected,
  show_selection,
  reset_selection,
  reload_global,
  refresh,
  config
)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
app_object	Cascade app object
filtered	barcodes to filter object
genes_to_plot	reactive list with genes in scratchpad
args	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
gene_choices	reactive list with all genes present in object
slice	reactive with slices to be used for plotting
all_selected	reactive containing list of selected points
show_selection	reactive to show selection
reset_selection	reactive to reset selection
reload_global	reactive to trigger reload
refresh	reactive to trigger plot refresh from sidebar button
config	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for spatial feature plot Server returns reactive expression containing selected points from the spatial feature plot

Examples

```

# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

spatial_coords <- data.frame(
  rn = colnames(obj),
  slice = as.character(metadata$orig.ident),
  imagecol = rep(c(1, 2, 1, 2), length.out = ncol(obj)),
  imagerow = rep(c(1, 1, 2, 2), length.out = ncol(obj)),
  stringsAsFactors = FALSE
)
imagerow_max <- as.list(
  tapply(spatial_coords$imagerow, spatial_coords$slice, max)
)
imagerow_min <- as.list(
  tapply(spatial_coords$imagerow, spatial_coords$slice, min)
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
  grouping_vars = grouping_vars,
  spatial_coords = spatial_coords,
  imagerow_max = imagerow_max,
  imagerow_min = imagerow_min
)

global_args <- list(
  assay = "RNA",
  slot = "data",

```

```

    grp_by = "cluster",
    dimred = "umap"
  )

  config <- get_config()

  ui <- shiny::fluidPage(
    shinyjs::useShinyjs(),
    shiny::sidebarLayout(
      shiny::sidebarPanel(spatialFeaturePlotUI("spatial_feature", "sidebar")),
      shiny::mainPanel(
        shiny::tabsetPanel(spatialFeaturePlotUI("spatial_feature", "main"),
          shiny::verbatimTextOutput("selected")
        )
      )
    )
  )

  server <- function(input, output, session) {
    selected <- spatialFeaturePlotServer(
      "spatial_feature",
      app_object = shiny::reactive({ app_object }),
      filtered = shiny::reactive({ colnames(obj) }),
      genes_to_plot = shiny::reactive({ "GeneA" }),
      args = shiny::reactive({ global_args }),
      gene_choices = shiny::reactive({ rownames(obj) }),
      slice = shiny::reactive({ unique(spatial_coords$slice)[1] }),
      all_selected = shiny::reactive({ list() }),
      show_selection = shiny::reactive({ NULL }),
      reset_selection = shiny::reactive({ NULL }),
      reload_global = shiny::reactiveVal(0),
      refresh = shiny::reactiveVal(0),
      config = shiny::reactive({ config })
    )

    output$selected <- shiny::renderPrint({
      selected()
    })
  }

  shiny::shinyApp(ui, server)

```

subsetServer

*Filter settings module server***Description**

Filter settings module server

Usage

```
subsetServer(id, obj, args, metadata_args, gene_choices, selected_points)
```

Arguments

id	Input id
obj	Cascade app object
args	reactive list with 'assay' with name of selected assay
metadata_args	reactive list with 'factor_levels' that has levels of categorical metadata & 'numeric_dist' that has distributions of numeric metadata
gene_choices	reactive list with all genes present in object
selected_points	reactive list with cell selections

Value

reactive expression containing filtered cell barcodes

subsetUI	<i>Filter settings module ui</i>
----------	----------------------------------

Description

Filter settings module ui

Usage

```
subsetUI(id)
```

Arguments

id	Input id
----	----------

Value

Shiny UI elements for the filter settings module

summarymod	<i>Summary tab module</i>
------------	---------------------------

Description

Summary tab module

Usage

```
summaryUI(id, panel)
```

```
summaryServer(id, obj, args)
```

Arguments

id	Input id
panel	string, can be 'sidebar' or 'main'
obj	Cascade app object
args	reactive with global arguments, 'project' & 'analysis'

Value

UI returns sidebar and main panel UI elements for summary module Server called for the side effect of rendering summary output.

Examples

```
obj <- make_example_seurat_object()

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = obj[[[]],
  qc = NULL
)

ui <- shiny::fluidPage(
  shiny::sidebarPanel(summaryUI("summary", "sidebar")),
  shiny::mainPanel(summaryUI("summary", "main"))
)

server <- function(input, output, session) {
  summaryServer(
    "summary",
    obj = app_object,
    args = shiny::reactive({
      list(
        project = file.path(tempdir(), "project1"),
```

```

        analysis = file.path(tempdir(), "project1", "analysis1", "object.rds")
      )
    })
  )
}

shiny::shinyApp(ui, server)

```

umap_ly

*Interactive umap plot***Description**

Interactive umap plot

Usage

```

umap_ly(
  df,
  xcol,
  ycol,
  color,
  colors,
  split = NULL,
  label_cols = NULL,
  showlegend = TRUE,
  showticklabels = TRUE,
  type = "scatter",
  highlight_click = TRUE,
  marker_size = 2,
  margin = 0.03,
  alpha = 0.3,
  free_axes = FALSE,
  width = NULL,
  height = NULL,
  source = "A"
)

```

Arguments

df	data.frame with plotting data
xcol	x-axis coordinates
ycol	y-axis coordinates
color	column to color markers by
colors	colors to use for markers

split	metadata column to split by
label_cols	Additional metadata columns to use for labeling pts
showlegend	should legend be shown?
showticklabels	should tick labels be drawn?
type	type of plotly trace (default: 'scatter')
highlight_click	should points be highlighted on clicking
marker_size	size of markers
margin	margin of plot (should be between 0 & 1)
alpha	marker opacity
free_axes	should subplots have free axes?
width	width of plot in pixels. If NULL (default), plot is auto-sized.
height	height of plot in pixels. if NULL (default), plot is auto-sized.
source	name of source to return data from

Value

plotly handle

Examples

```
df <- data.frame(
  UMAP_1 = c(-1, 0, 1, 2),
  UMAP_2 = c(0, 1, 0, -1),
  cluster = factor(c("A", "A", "B", "B"))
)
colors <- c(A = "#4477aa", B = "#cc6677")

umap_ly(
  df,
  xcol = "UMAP_1",
  ycol = "UMAP_2",
  color = "cluster",
  colors = colors
)
```

violin_ly

Interactive violin plot

Description

Violin plot implemented in plotly. This works fine, but is a bit slower/takes the same time as the ggplot version. But the main issue is that combining multiple plots using subplot() results in very skinny violins.

- This might be a good option for single gene plots.

Usage

```
violin_ly(df, xcol, ycol, color, colors, showlegend = FALSE)
```

Arguments

df	data.frame with expression data
xcol	x-axis grouping column
ycol	values to show on y-axis
color	column to split violins by
colors	vector of colors to use (not used)
showlegend	boolean, should legend be shown? Default: FALSE

Details

NOTE: not used

Value

plotly handle

violin2	<i>Violin plot</i>
---------	--------------------

Description

Lightweight reimplementation of violin plot with additional splitting features. This works on just a data frame and does not need to be passed the entire object.

Usage

```
violin2(  
  df,  
  xcol,  
  ycol,  
  color = NULL,  
  colors = NULL,  
  draw_points = FALSE,  
  scales = "fixed",  
  text_scale = 1  
)
```

Arguments

<code>df</code>	data.frame with expression data
<code>xcol</code>	x-axis grouping column
<code>ycol</code>	values to show on y-axis
<code>color</code>	column to split violins by
<code>colors</code>	vector of colors to use (not used)
<code>draw_points</code>	show points for each cell? Default: FALSE
<code>scales</code>	should y-axis be 'fixed' (default) or 'free'
<code>text_scale</code>	scaling factor for text labels. If < 1, text size is reduced and vice-versa

Value

ggplot2 handle

Examples

```
df <- data.frame(
  cluster = factor(rep(c("A", "B"), each = 4)),
  variable = rep(c("GeneA", "GeneB"), times = 4),
  value = c(0, 2, 4, 5, 1, 3, 6, 8)
)

violin2(df, xcol = "cluster", ycol = "value")
```

violinmod

Violin plot module

Description

Violin plot module

Usage

```
violinUI(id, panel)

violinServer(
  id,
  app_object,
  filtered,
  genes_to_plot,
  args,
  gene_choices,
  reload_global,
  refresh,
  config
)
```

Arguments

<code>id</code>	Input id
<code>panel</code>	string, can be 'sidebar' or 'main'
<code>app_object</code>	Cascade app object
<code>filtered</code>	barcodes to filter object
<code>genes_to_plot</code>	reactive list with genes in scratchpad
<code>args</code>	reactive list with elements: 'assay' for selected assay, 'dimred' for which dimension reduction to use and 'grp_by' for grouping variable
<code>gene_choices</code>	reactive list with all genes present in object
<code>reload_global</code>	reactive to trigger reload
<code>refresh</code>	reactive to trigger plot refresh from sidebar button
<code>config</code>	reactive list with config settings

Value

UI returns sidebar/main panel UI elements for violin plot Server called for the side effect of rendering a violin plot.

Examples

```
# example obj
obj <- make_example_seurat_object()

# prep metadata
metadata <- obj[[[]]]
metadata_levels <- lapply(
  metadata[c("cluster", "condition", "orig.ident", "seurat_clusters")],
  levels
)

# get grouping vars and colors
grouping_vars <- names(metadata_levels)
names(grouping_vars) <- paste0(
  grouping_vars,
  " (n = ",
  lengths(metadata_levels),
  ")"
)

cluster_colors <- lapply(metadata_levels, function(lvls) {
  stats::setNames(rep_len(c("#4477aa", "#cc6677"), length(lvls)), lvls)
})

app_object <- list(
  rds = obj,
  obj_type = "seurat",
  metadata = metadata,
  metadata_levels = metadata_levels,
  cluster_colors = cluster_colors,
```

```
      grouping_vars = grouping_vars,
      spatial_coords = NULL,
      imagerow_max = NULL,
      imagerow_min = NULL
    )

    global_args <- list(
      assay = "RNA",
      slot = "data",
      grp_by = "cluster",
      dimred = "umap"
    )

    config <- get_config()

    ui <- shiny::fluidPage(
      shiny::sidebarLayout(
        shiny::sidebarPanel(violinUI("violin", "sidebar")),
        shiny::mainPanel(shiny::tabsetPanel(violinUI("violin", "main")))
      )
    )

    server <- function(input, output, session) {
      violinServer(
        "violin",
        app_object = shiny::reactive({ app_object }),
        filtered = shiny::reactive({ colnames(obj) }),
        genes_to_plot = shiny::reactive({ c("GeneA", "GeneB") }),
        args = shiny::reactive({ global_args }),
        gene_choices = shiny::reactive({ rownames(obj) }),
        reload_global = shiny::reactiveVal(0),
        refresh = shiny::reactiveVal(0),
        config = shiny::reactive({ config })
      )
    }

    shiny::shinyApp(ui, server)
```

Index

add_cascade_analysis, 3

BlendExpression2, 5

BlendMap2, 5

BuildClusterTree2, 6

check_user_access, 6

clustreemod, 7

clustreeServer (clustreemod), 7

clustreeUI (clustreemod), 7

clustsummarymod, 9

clustSummaryServer (clustsummarymod), 9

clustSummaryUI (clustsummarymod), 9

coexpmod, 11

coexpressionPlotServer (coexpmod), 11

coexpressionPlotUI (coexpmod), 11

controlmod, 14

controlServer (controlmod), 14

controlUI (controlmod), 14

create_access_yaml, 15

dimredmod, 16

dimredServer (dimredmod), 16

dimredUI (dimredmod), 16

dlfilemod, 18

dlmod, 19

dotplot, 19

dotplotmod, 20

dotPlotServer (dotplotmod), 20

dotPlotUI (dotplotmod), 20

downloadFileServer (dlfilemod), 18

downloadFileUI (dlfilemod), 18

downloadPlotServer (dlmod), 19

downloadPlotUI (dlmod), 19

feature_blend, 22

feature_ly, 24

featurePlotServer (ftrpltmod), 26

featurePlotUI (ftrpltmod), 26

ftrpltmod, 26

get_access_path, 29

get_binned_exp, 29

get_coexp_legend, 30

get_coexp_tbl, 31

get_coexplt_colors, 32

get_config, 32

get_label_trace, 33

get_limits, 33

get_project_name_from_path, 34

get_var_genes, 35

helpButtonServer (helpmod), 35

helpButtonUI (helpmod), 35

helpmod, 35

helpModal, 36

install_cascade, 37

linemod, 37

linePlotServer (linemod), 37

linePlotUI (linemod), 37

make_example_seurat_object, 38

markerPlotServer (markerpltmod), 39

markerPlotUI (markerpltmod), 39

markerpltmod, 39

markertablegeneralmod, 41

markerTableGeneralServer (markertablegeneralmod), 41

markerTableGeneralUI (markertablegeneralmod), 41

markertablemod, 44

markerTableServer (markertablemod), 44

markerTableUI (markertablemod), 44

PseudoBulkExpression2, 45

read_access_yaml, 46

run_cascade, 47

sanitize_colnames, 47

save_access_yaml, [48](#)
scattermod, [49](#)
scatterPlotServer (scattermod), [49](#)
scatterPlotUI (scattermod), [49](#)
selectionServer (selmod), [51](#)
selectionUI (selmod), [51](#)
selmod, [51](#)
settingsmod, [52](#)
settingsServer (settingsmod), [52](#)
settingsUI (settingsmod), [52](#)
spatialcoexpmod, [53](#)
spatialCoexpressionPlotServer
 (spatialcoexpmod), [53](#)
spatialCoexpressionPlotUI
 (spatialcoexpmod), [53](#)
spatialFeaturePlotServer
 (spatialftrpltmod), [56](#)
spatialFeaturePlotUI
 (spatialftrpltmod), [56](#)
spatialftrpltmod, [56](#)
subsetServer, [59](#)
subsetUI, [60](#)
summarymod, [61](#)
summaryServer (summarymod), [61](#)
summaryUI (summarymod), [61](#)

umap_ly, [62](#)

violin2, [64](#)
violin_ly, [63](#)
violinmod, [65](#)
violinServer (violinmod), [65](#)
violinUI (violinmod), [65](#)