Security and Scalability in Shiny with {httr2}

Strategies for Efficient API Use

Alexandros Kouretsis, PhD

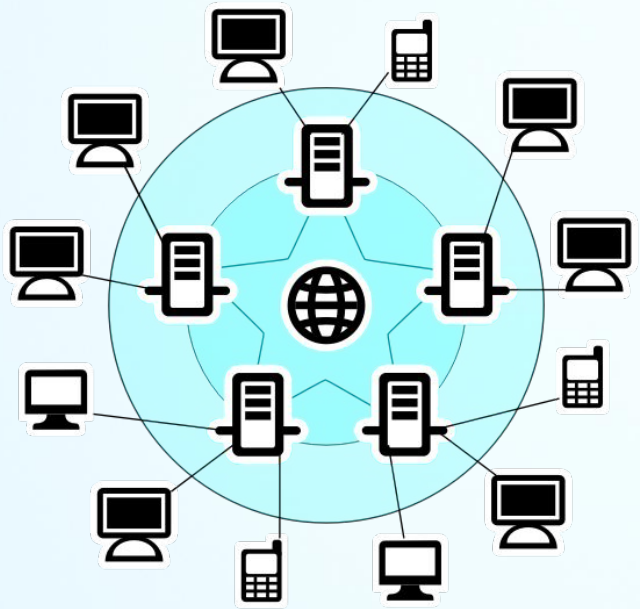
Appsilon

Understanding Web APIs

"Allow different software applications to communicate over the internet"

Understanding Web APIs

"Allow different software applications to communicate over the internet"



Data Structures

XML

JSON

RESTful Architecture

Stateless Communication

RESTful Architecture

- Stateless Communication
- CRUD (create, read, update, del.)

RESTful Architecture

- Stateless Communication
- CRUD (create, read, update, del.)
- Developer Ecosystem

Challenges

Structuring Requests

Challenges

- Structuring Requests
- Response Handling

Challenges

- Structuring Requests
- Response Handling
- Security

Challenges

- Structuring Requests
- Response Handling
- Security
- Optimisation, Test, Rate Limits

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: length_here

{
  "userId": "123",
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: length_here

{
  "userId": "123",
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

GET request example

```
GET /api/data HTTP/1.1
Host: example.com
Authorization: Bearer your_access_token_here
Accept: application/json
```

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: length_here

{
  "userId": "123",
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

User Friendly Functions

User Friendly Functions

Configuration of Headers

User Friendly Functions

Authentication

Configuration of Headers

User Friendly Functions

Authentication

Configuration of Headers

Response Handling

User Friendly Functions

Authentication

Configuration of Headers

Response Handling

Errors and Rate Limits

Tidyverse compatible

```
request <- request("https://api.example.com") |>
  req_url("/data") |>
  req_url_query(key = "value") |>
  req_progress() |>
  req_perform()
```

Composable

```
request <- request("https://api.example.com") |>
  req_url("/data") |>
  req_progress()

response <- request() |>
  req_url_query(key = "value") |>
  req_perform()
```

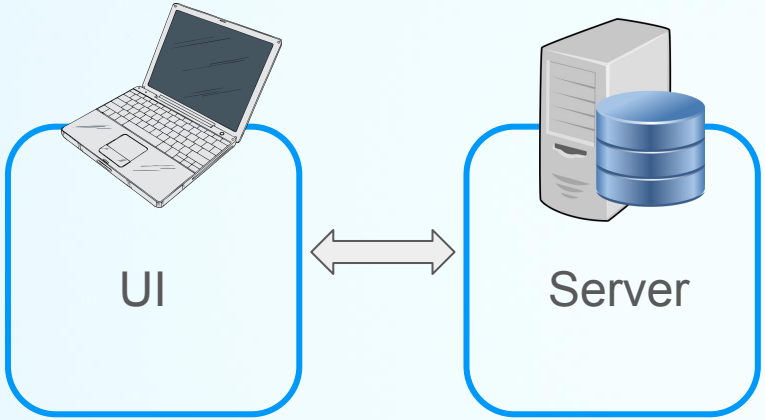
Composable

```
request <- request("https://api.example.com") |>
  req_url("/data") |>
  req_progress()

response <- request() |>
  req_url_query(key = "value") |>
  req_perform()
```

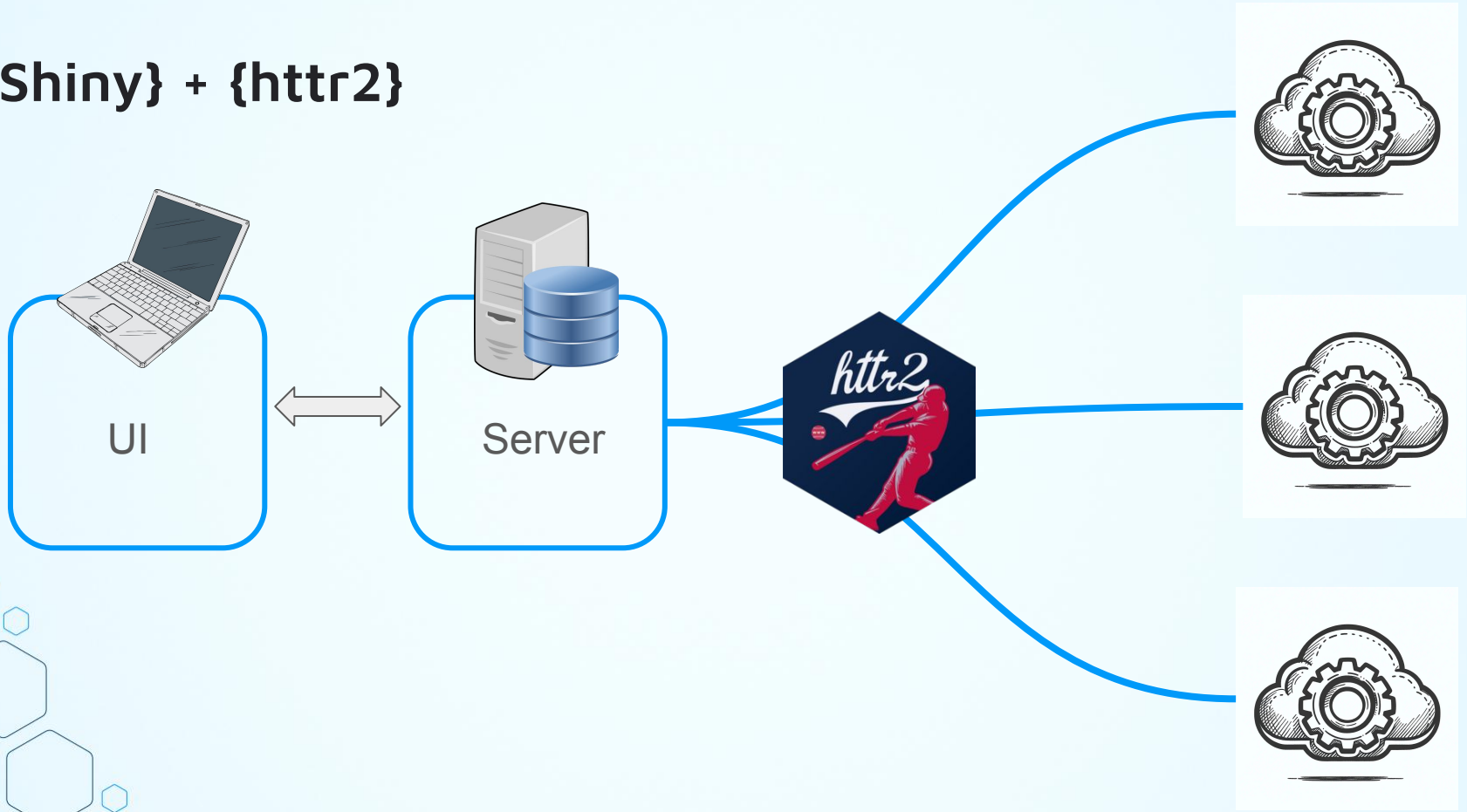
Composable

```
request <- request("https://api.example.com") |>
  req_url("/data") |>
  req_progress()

response <- request() |>
  req_url_query(key = "value") |>
  req_perform()
```

{Shiny} + {httr2}



{Shiny} + {httr2}



Code structure tips

“Use Centralised approach per session”

Code structure tips

"Use Centralised approach per session"

1 Single Responsibility

Code structure tips

"Use Centralised approach per session"

1. Single Responsibility
2. Error Handling

Code structure tips

"Use Centralised approach per session"

1. Single Responsibility
2. Error Handling
3. Maintenance & Updates

Code structure tips

```
server <- function(input, output) {
  session$userData$my_api <- myApi$new()

  ...

}
```

Code structure tips

```
server <- function(input, output) {
  session$userData$my_api <- myApi$new()

  ...

}
```

```
data <- reactive(
  session$userData$my_api$get_data(...)
)
```

Manage API calls

Stateful Architecture

Manage API calls

- Stateful Architecture
- Cache API calls ⚠️🔒

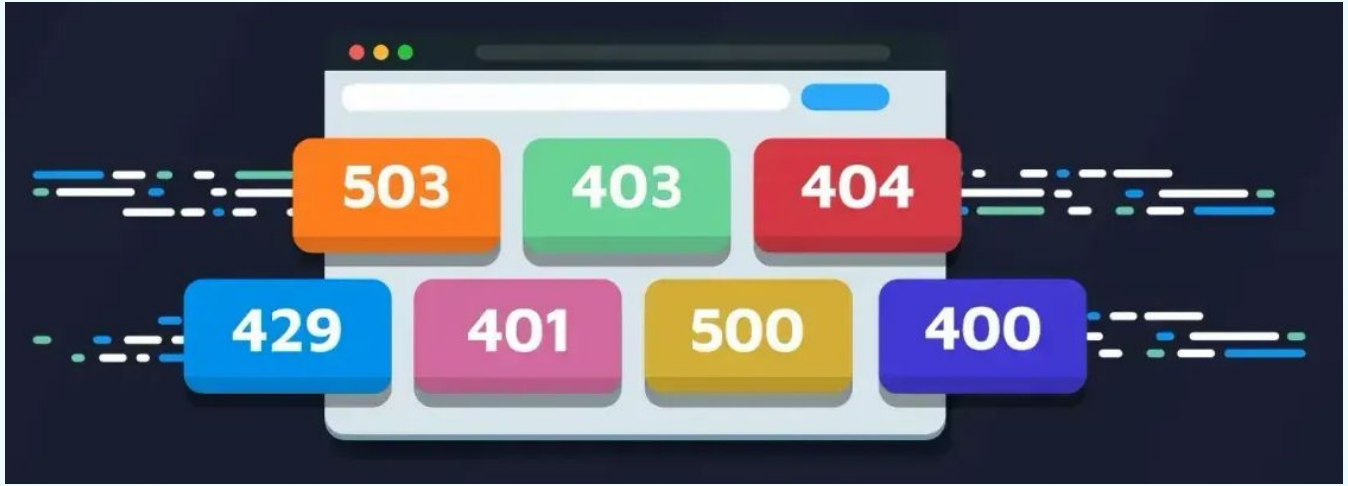
Manage API calls

- Stateful Architecture
- Cache API calls ⚠️🔒
- CRUD in `observe(...)`

Manage API calls

- Stateful Architecture
- Cache API calls ⚠️🔒
- CRUD in `observe(...)`
- Calc. in `reactive(...)`

Error Handling



Error Handling

Request Errors

Error Handling

Request Errors

Transient Errors

Error Handling

```
observe({
  tryCatch({
    session$userData$my_api$get_data(key = input$value)
  }, error_400 = function(e) {
    showNotification(e$messafe, type = "error")
  })
})
```

Error Handling

```
observe({
  tryCatch({
    session$userData$my_api$get_data(key = input$value)
  }, error_400 = function(e) {
    showNotification(e$messafe, type = "error")
  })
})
```

Error Handling

```
observe({
  tryCatch({
    session$userData$my_api$get_data(key = input$value)
  }, error_400 = function(e) {
    showNotification(e$messafe, type = "error")
  })
})
```

```
rlang::abort("error400msg", class = "error_400")
```

Transient Errors

```
request("http://google.com") |>
  req_retry(is_transient = \(resp) resp_status(resp) %in% c(429, 500, 503))
```

Transient Errors

```
request("http://google.com") |>
  req_retry(is_transient = \(resp) resp_status(resp) %in% c(429, 500, 503))
```

Transient Errors

```
request("http://google.com") |>
  req_retry(is_transient = \(resp) resp_status(resp) %in% c(429, 500, 503))
```

```
tryCatch(
  withCallingHandlers(
    session$userData$my_api$get_data(key = input$value),
    message = function(m) showNotification(m$message, type = "message"),
    warning = function(w) showNotification(w$message, type = "warning")
  ),
  error_400 = function(e) showNotification(e$message, type = "error")
)
```

Authentication

`req_auth_basic()`
Authenticate request with HTTP basic authentication

`req_auth_bearer_token()`
Authenticate request with bearer token

`req_oauth_auth_code()` `oauth_flow_auth_code()`
OAuth with authorization code

`req_oauth_bearer_jwt()` `oauth_flow_bearer_jwt()`
OAuth with a bearer JWT (JSON web token)

`req_oauth_client_credentials()` `oauth_flow_client_credentials()`
OAuth with client credentials

`req_oauth_device()` `oauth_flow_device()`
OAuth with device flow

`req_oauth_password()` `oauth_flow_password()`
OAuth with username and password

`req_oauth_refresh()` `oauth_flow_refresh()`
OAuth with a refresh token

Improved Security

“Reduced client side exposure”

Stateful Architecture

Server side API calls

Control at inputs

Improved Security

“Reduced client side exposure”

Token theft
Replay attacks
Excessive Data Exposure
Mass assignment

Extended Task 🥳🥳🥳

🔄 Processing...

Extended Task 🥳🥳🥳

```
fetch_data <- ExtendedTask$new(function(x) {
  future_promise({
    my_api$get_data(key = x)
  })
}) |> bind_task_button("btn")

observeEvent(input$btn, {
  fetch_data$invoke(input$value)
})
```

Extended Task 🥳🥳🥳

```
fetch_data <- ExtendedTask$new(function(x) {
  future_promise({
    my_api$get_data(key = x)
  })
}) |> bind_task_button("btn")

observeEvent(input$btn, {
  fetch_data$invoke(input$value)
})
```

Security

Stability

Performance

Thank you!

Alexandros Kouretsis

Appsilon