

SOLID Elixir

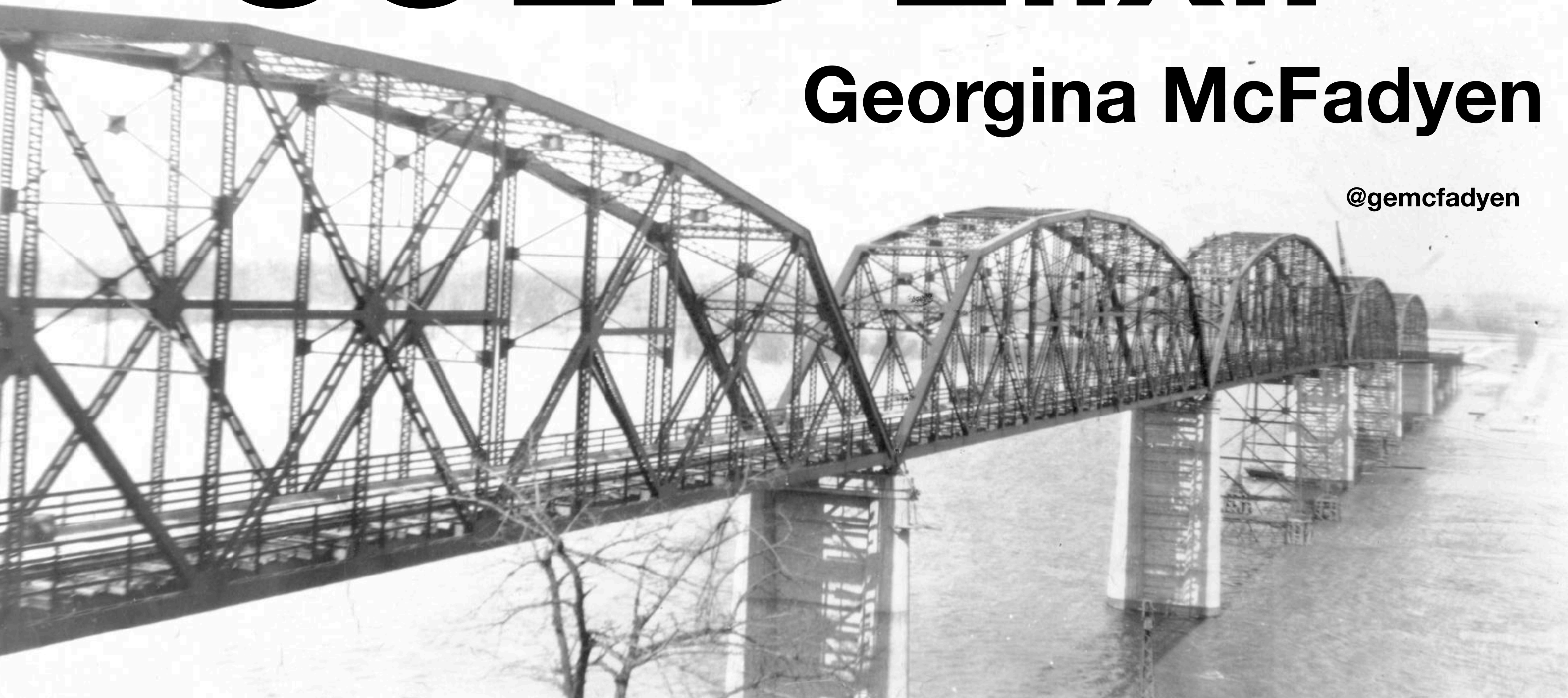
Georgina McFadyen



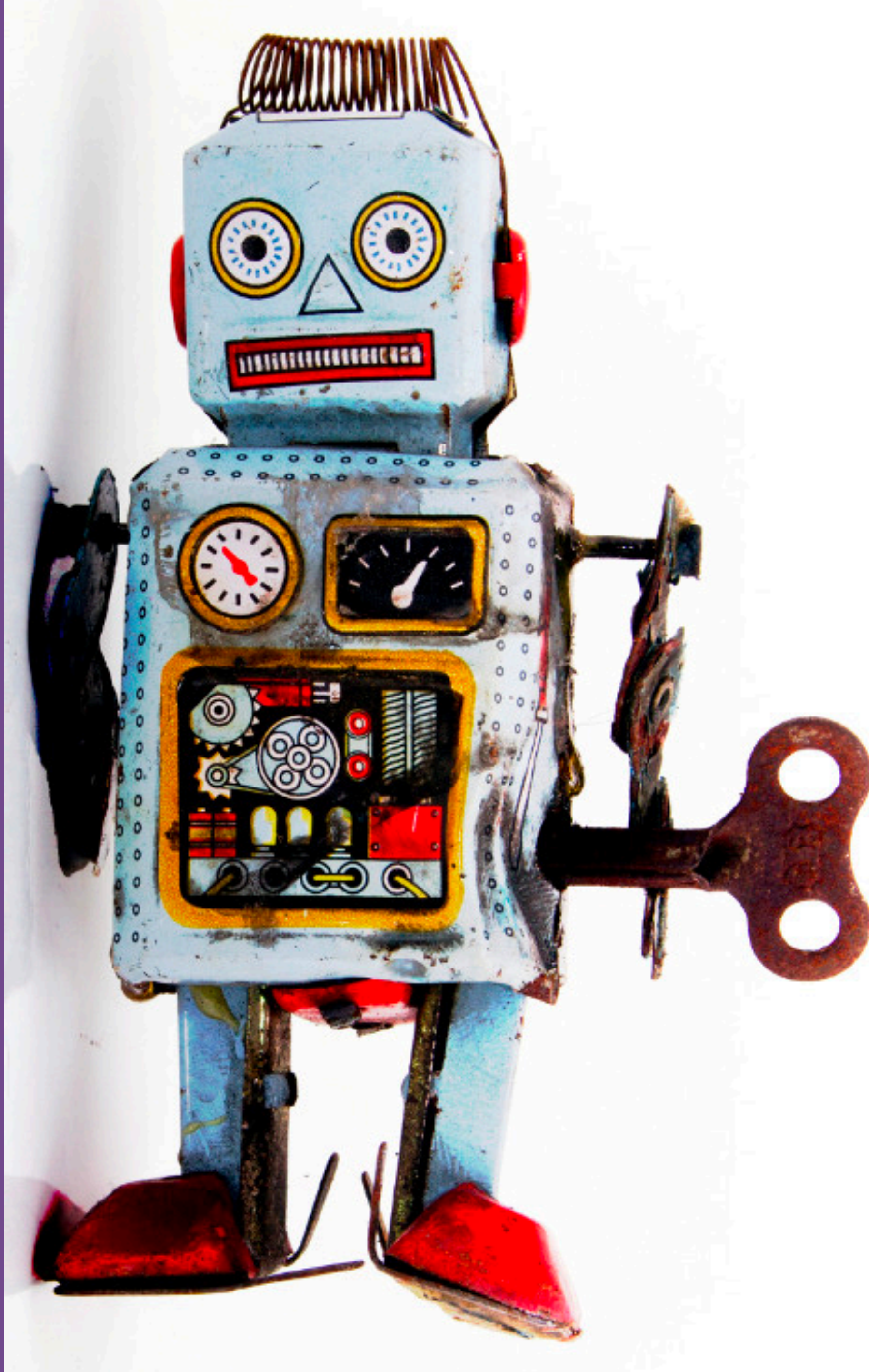
SOLID Elixir

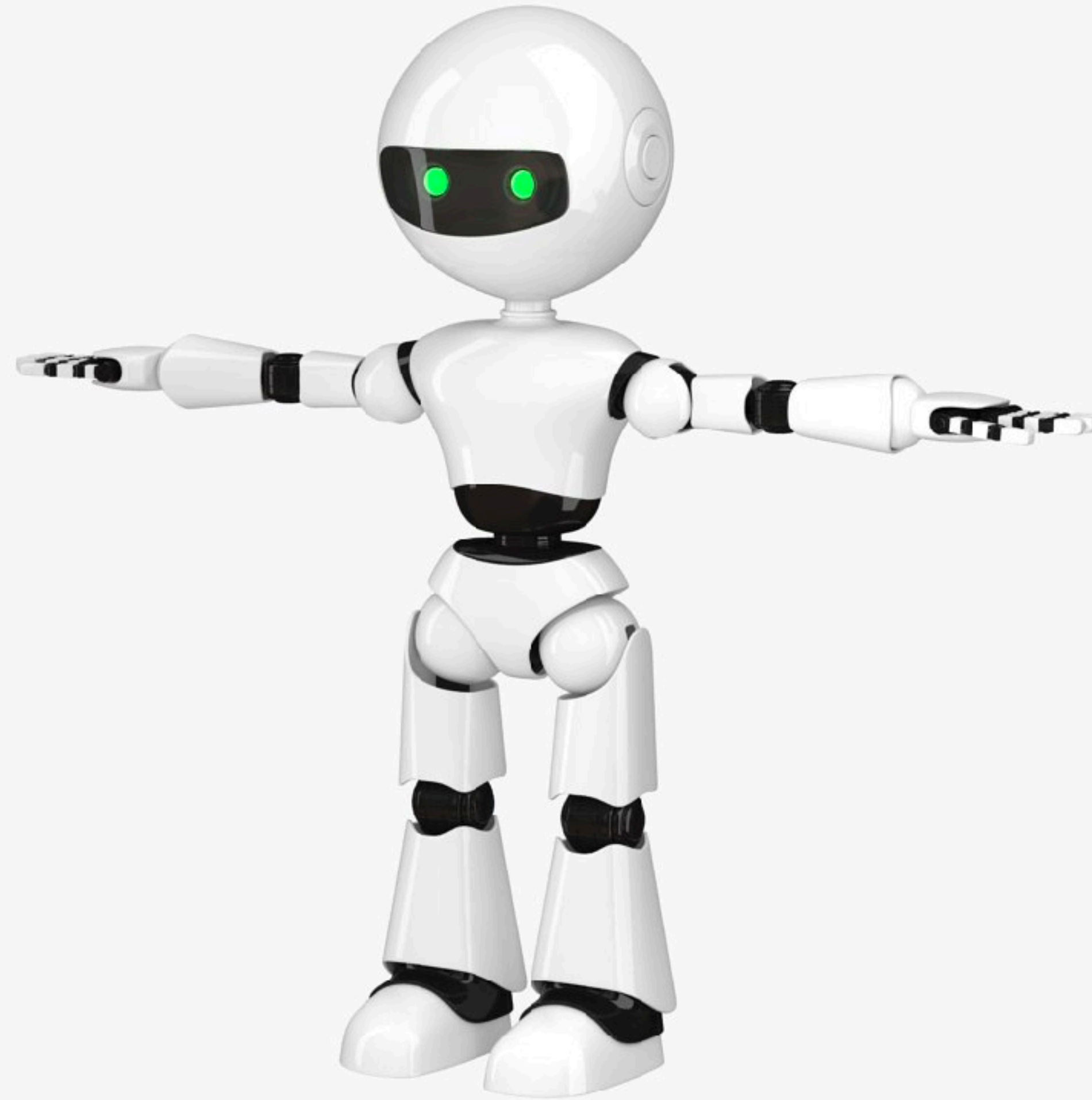
Georgina McFadyen

@gemcfadyen



Hi, I'm Georgina







**What are the SOLID
principles?**

“In Object-Oriented programming,
the term SOLID is a mnemonic
acronym for five design principle
intended to make software design
more understandable flexible and
maintainable .”

– *Wikipedia*

[https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design))

Holiday Booking Domain

https://github.com/gemcfadyen/solid_elixir



Select your accommodation preferences

Hotel

Apartment

Guest House

Bedrooms

☐ 1

☒ 2

☐ 3

☐ 4

Catering

☒ Self-Catering

☐ All Inclusive

Parking

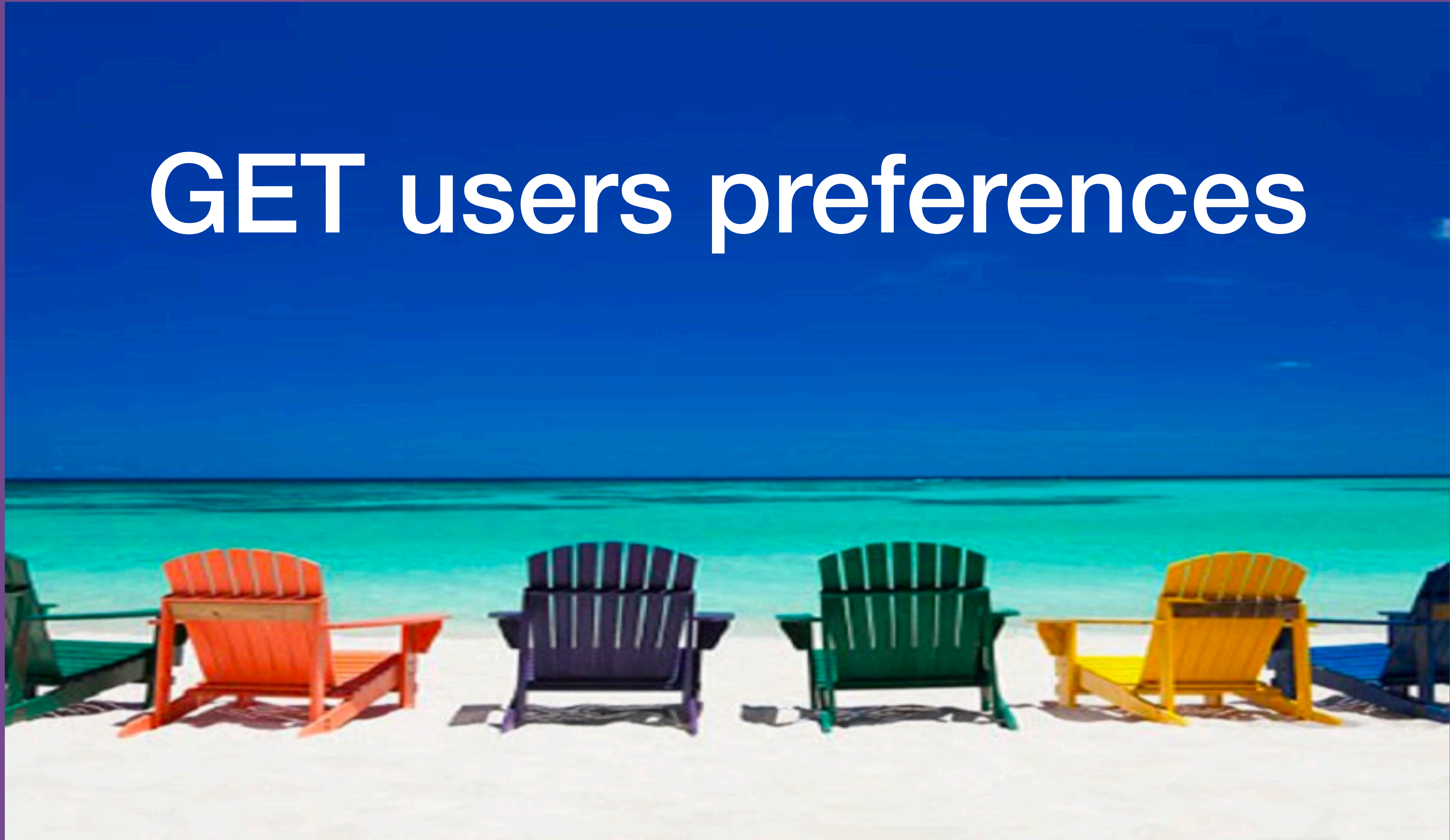
☐ None

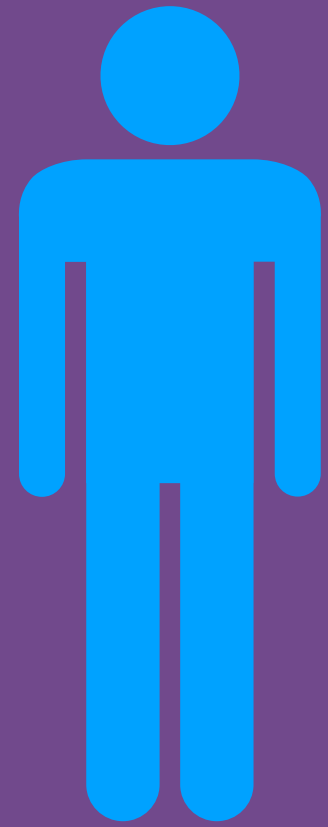
☐ On Street

☒ Secure Garage

Search

GET users preferences





Login



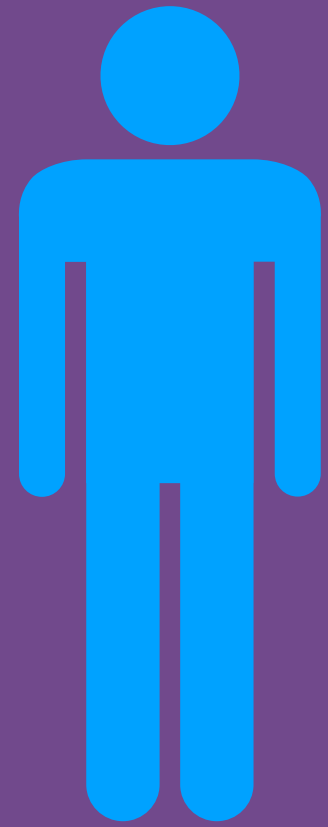
Select your accommodation preferences

Hotel	Apartment	Guest House
-------	-----------	-------------

Bedrooms	Catering	Parking
☐ 1	☒ Self-Catering	☐ None
☒ 2	☐ All Inclusive	☐ On Street
☐ 3		☒ Secure Garage
☐ 4		

Search

GET users preferences



Login



Select your accommodation preferences

Hotel	Apartment	Guest House
-------	-----------	-------------

Bedrooms	Catering	Parking
☐ 1	☒ Self-Catering	☐ None
☒ 2	☐ All Inclusive	☐ On Street
☐ 3		☒ Secure Garage
☐ 4		

Search



id of customer

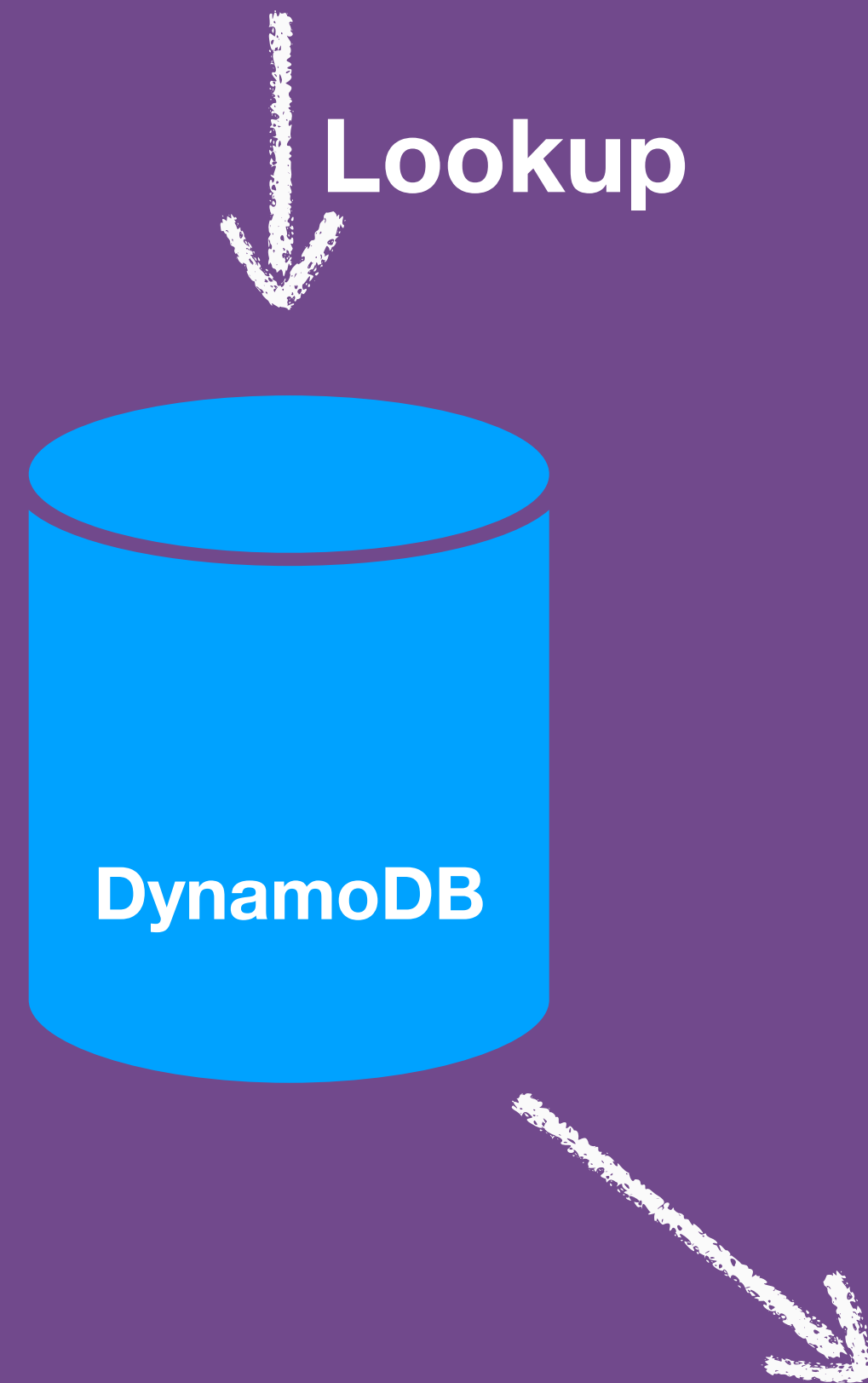
GET /customers/{id}

GET /customers/{id}

↓ Lookup



GET /customers/{id}

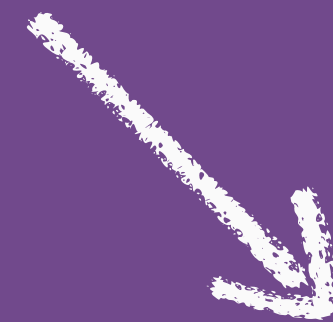


```
%{"id" => "uuid-1",  
  "lastModified" => 1519928745635,  
  "preferences" =>  
    %{"accommodation" =>  
      %{"apartment" => %{"  
        "catering" => "self_catering",  
        "bedrooms" => 2,  
        "parking" => "secure"  
      }},  
      "hotel" => %{"  
        "catering" => "self_catering",  
        "bedrooms" => 1,  
        "parking" => "secure"  
      }  
    }  
}
```

GET /customers/{id}

```
%{"id" => "uuid-1",  
  "lastModified" => 1519928745635,  
  "preferences" =>  
    %{"accommodation" =>  
      %{"apartment" => %{"  
        "catering" => "self_catering",  
        "bedrooms" => 2,  
        "parking" => "secure"  
      },  
      "hotel" => %{"  
        "catering" => "self_catering",  
        "bedrooms" => 1,  
        "parking" => "secure"  
      }  
    }  
}
```

Internal Struct Representation

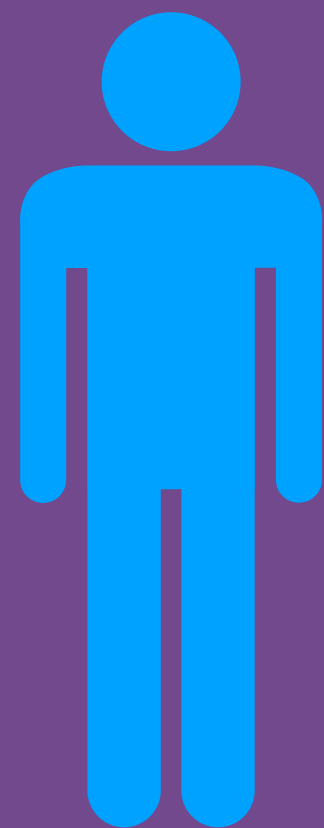


```
{  
  "id": "uuid-1",  
  "lastModified": "2018-02-21T12:05:15Z",  
  "preferences": {  
    "accommodation": {  
      "apartment": {  
        "catering": "self_catering",  
        "bedrooms": 2,  
        "parking": "secure"  
      },  
      "hotel": {  
        "catering": "self_catering",  
        "bedrooms": 1,  
        "parking": "secure"  
      }  
    }  
  }  
}
```

JSON Representation

POST users preferences





Login



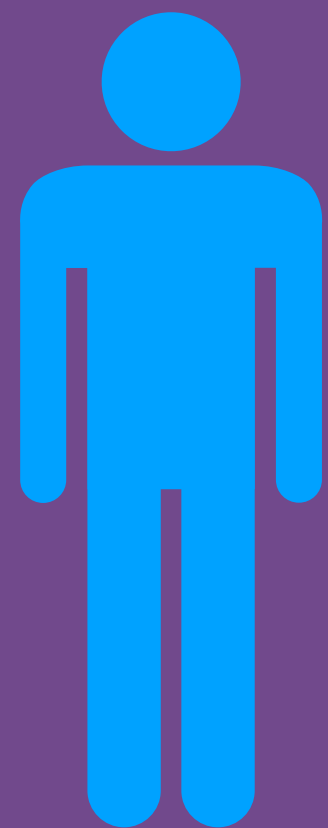
Select your accommodation preferences

Hotel	Apartment	Guest House
-------	-----------	-------------

Bedrooms	Catering	Parking
☐ 1	☒ Self-Catering	☐ None
☒ 2	☐ All Inclusive	☐ On Street
☐ 3		☒ Secure Garage
☐ 4		

Search

POST /customers



Login



Search



Select your accommodation preferences

Hotel	Apartment	Guest House
-------	-----------	-------------

Bedrooms	Catering	Parking
☐ 1	☒ Self-Catering	☐ None
☒ 2	☐ All Inclusive	☐ On Street
☐ 3		☒ Secure Garage
☐ 4		

Search



POST /customers

POST /customers

```
{
  "id": "uuid-1",
  "lastModified": "2018-02-21T12:05:15Z",
  "preferences": {
    "accommodation": {
      "apartment": {
        "catering": "self_catering",
        "bedrooms": 2,
        "parking": "secure"
      },
      "hotel": {
        "catering": "self_catering",
        "bedrooms": 1,
        "parking": "secure"
      }
    }
  }
}
```

JSON Representation

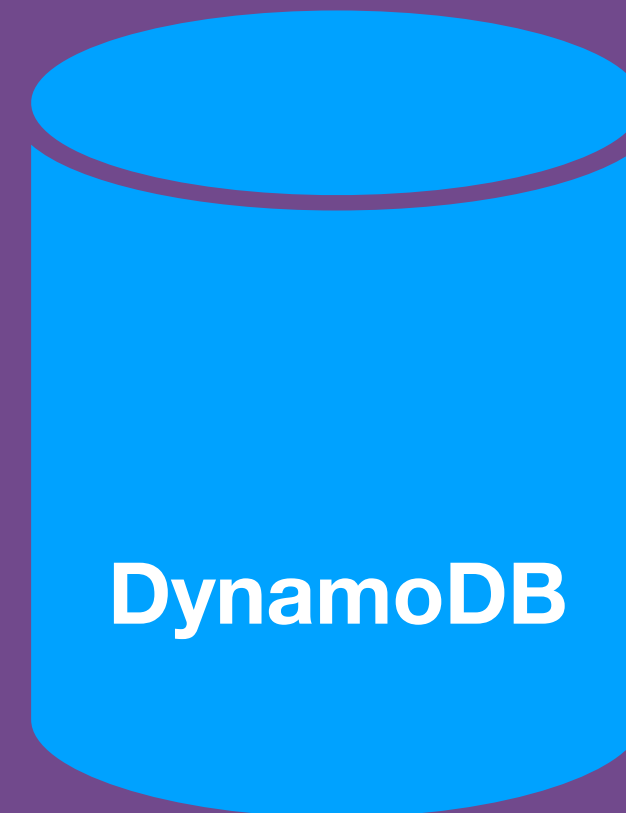


```
%{"id" => "uuid-1",
  "lastModified" => 1519928745635,
  "preferences" =>
    %{"accommodation" =>
      %{"apartment" => %{
        "catering" => "self_catering",
        "bedrooms" => 2,
        "parking" => "secure"
      },
      "hotel" => %{
        "catering" => "self_catering",
        "bedrooms" => 1,
        "parking" => "secure"
      }
    }
}
```

Internal Struct Representation

POST /customers

↓ Persist



SOLID

Single Responsibility

Open Close

Liskov Substitution

Interface Segregation

Dependency Inversion





Single Responsibility

SOLID

“The single responsibility principle states that every **module** or **class** should have **responsibility over a single part** of the functionality provided by the software...”

– *Wikipedia*

https://en.wikipedia.org/wiki/Single_responsibility_principle

```

1 ▼ defmodule Controllers.PreferenceController do
2
3 ▼   def get_preferences(customer_id) do
4     ExAws.Dynamo.get_item("customer-preferences", %{id: customer_id})
5     |> ExAws.request
6     |> decode
7   end
8   defp decode({:ok, response}) when map_size(response) > 0 do
9     {:error, :not_found}
10  end
11 ▼   defp decode({:ok, response}) do
12     valid_consent_user = ExAws.Dynamo.decode_item(response,
13       schema: Schema.Database.CustomerPreferencesRow)
14     |> format_response()
15
16     {:ok, valid_consent_user}
17   end
18   defp decode({:error, response}) do
19     {:error, response}
20   end
21
22 ▼   def format_response(raw_response) do
23     pretty_response = raw_response
24     # Data formatting to create a nice response for calling system
25     pretty_response
26   end
27 end

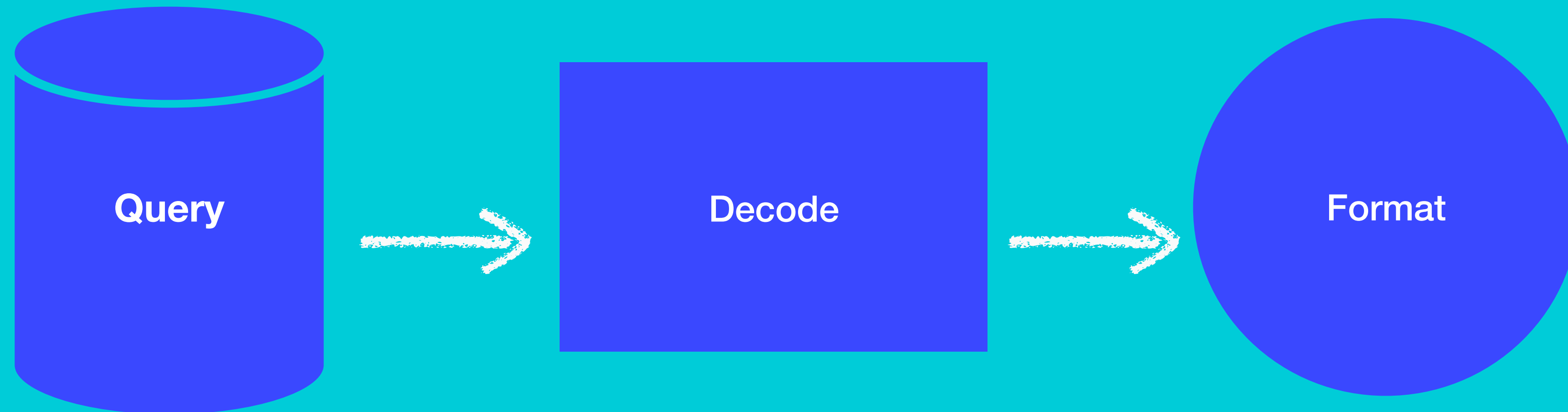
```

Fetch User Preferences

GET

Controller Responsibilities

defmodule PreferenceController



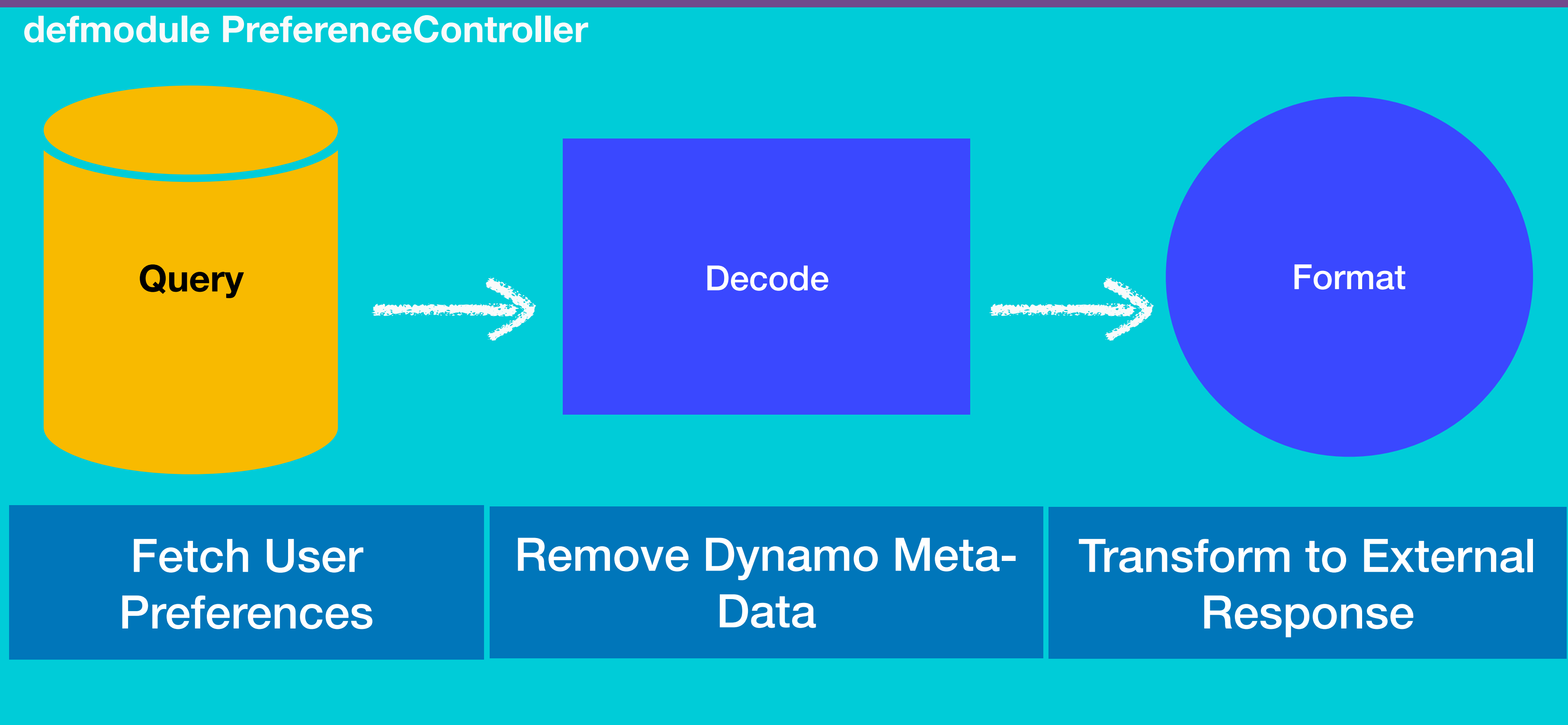
Fetch User
Preferences

Remove Dynamo Meta-
Data

Transform to External
Response

Can any of these steps
change independently?

What if we changed the conditions on which we query?



“**Robert C. Martin** expresses
the principle as, "A class
should have only one reason
to change.”

– *Wikipedia*

https://en.wikipedia.org/wiki/Single_responsibility_principle

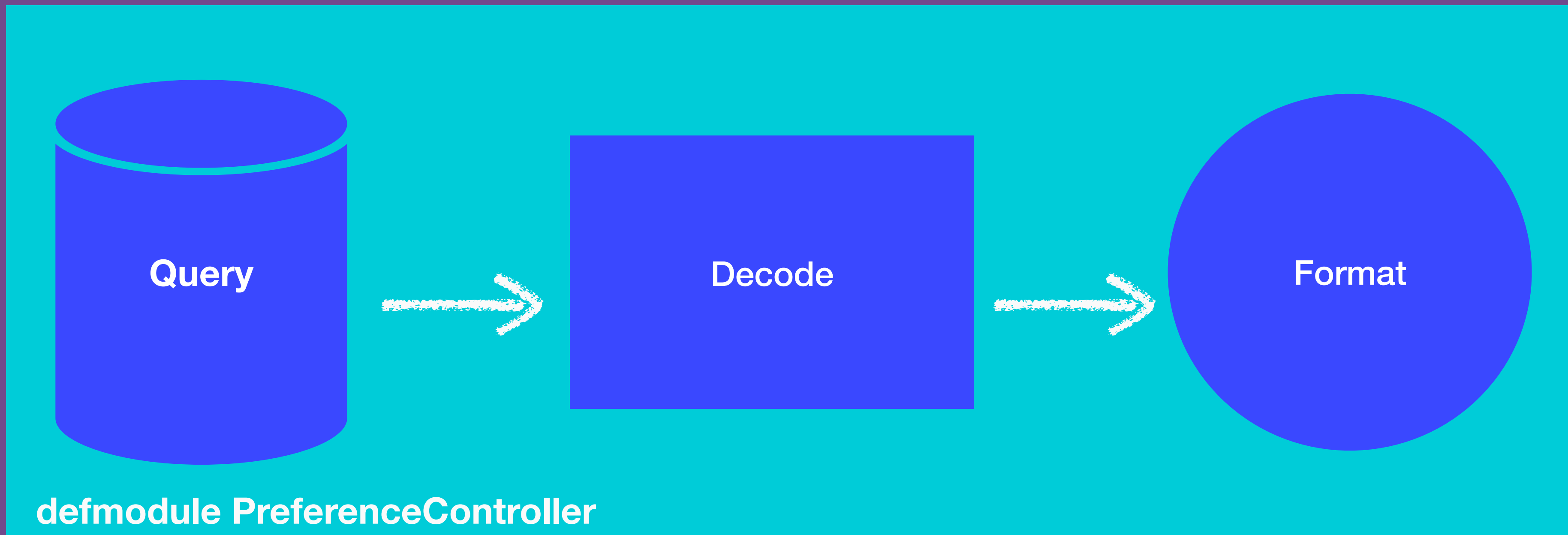
“**Robert C. Martin** expresses the principle as, "A class should have only one reason to change.”

“module”

– *Wikipedia*

https://en.wikipedia.org/wiki/Single_responsibility_principle

Controller Responsibilities



Query Database

```
defmodule Core.Preferences.GetCustomerPreferences do

  def for(customer_id) do
    ExAws.Dynamo.get_item("customer-preferences",
                          %{id: customer_id})
    |> ExAws.request
  end
end
```



Decode Database Response

```

1 defmodule Core.Preferences.Decode do
2
3   def decode({:ok, response}) when map_size(response) == 0 do
4     {:error, :not_found}
5   end
6   defp decode({:ok, response}) do
7     valid_preferences = ExAws.Dynamo.decode_item(response,
8       |>|>|>|>|>|> as: Schema.Database.CustomerPreferencesRow)
9
10    {:ok, valid_prefereneces}
11  end
12  defp decode({:error, response}) do
13    {:error, response}
14  end
15 end

```

Decode

Decode Database Response

```
1 defmodule Core.Preferences.Decode do
2
3   def decode({:ok, response}) when map_size(response) == 0 do
4     {:error, :not_found}
5   end
6   defp decode({:ok, response}) do
7     valid_preferences = ExAws.Dynamo.decode_item(response,
8       |> as: Schema.Database.CustomerPreferencesRow)
9
10    {:ok, valid_preferences}
11  end
12  defp decode({:error, response}) do
13    {:error, response}
14  end
15 end
```

Decode

Format to Json Response

```
1 ▾ defmodule Core.Schema.External.Response do
2 ▾   def format({:ok, database_response}) do
3     database_response
4     |> format_dates()
5     |> group_accomodation_preferences()
6   end
7
8   defp format_dates(raw_data) do
9     ..
10  end
11
12  defp group_accomodation_preferences(raw_data) do
13    ...
14  end
15 ▾ end
```

Format

Controller

```
1 defmodule Controllers.PreferenceController do
2
3   def get_preferences(customer_id) do
4     Core.Preferences.GetCustomerPreferences.for(customer_id)
5     |> Core.Schema.External.Response.format()
6   end
7 end
```


Benefits of Single Responsibility

SOLID

- ✓ Focused unit tests
- ✓ Fewer higher level tests
- ✓ Less dependencies per test
- ✓ Similar functionality grouped

Trade Off

 Ensure each module justifies it's
existence

Open Close SOLID



“Software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification**...An entity can allow its behaviour to be **extended without modifying its source code**”

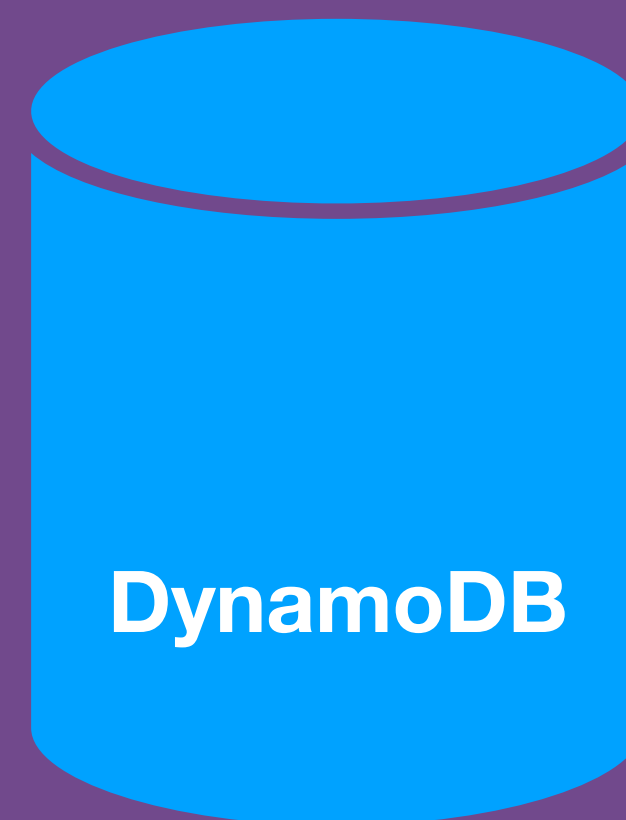
– *Wikipedia*

https://en.wikipedia.org/wiki/Open/closed_principle

Open Closed

Ability to add new code
without modifying any
existing code.

POST /customers



POST /customers

validate
request



Persist

DynamoDB

Request Validation

- ? Is query parameter a valid uuid?
- ? Does body contain preferences section?
- ? Does body contain accommodation section?

```

1  defmodule Plugs.SaveCustomer do
2    import Plug.Conn
3
4    def init(opts) do
5      opts
6    end
7
8    def call(conn, _opts) do
9      ...
10     with {:ok, customer_id} <-
11       Validation.RequestParameterValidation
12         .has_valid_parameters(conn.path_params),
13       :ok <-
14         Validation.PreferencesValidation
15           .has_preferences(body),
16       :ok <-
17         Validation.AccommodationValidation
18           .has_accommodation_preferences(body) do
19       ...
20     end
21   end
22 end
23 end
24 end
25 end
26 end
27 end
28 end
29 end
30 end
31 end

```

validate different parts of request


```

1  defmodule Plugs.SaveCustomer do
2    import Plug.Conn
3
4    def init(opts) do
5      opts
6    end
7
8    def call(conn, _opts) do
9      body = conn.body_params
10
11      with {:ok, customer_id} <-
12        Validation.RequestParameterValidation.has_valid_parameters(conn.path_params),
13         :ok <-
14           Validation.PreferencesValidation.has_preferences(body),
15         :ok <-
16           Validation.AccommodationValidation.has_accommodation_preferences(b
17

```

Save Preferences

```

18  valid_request = Schema.Http.Request.new!(body,
19                                           customer_id,
20                                           body["version"])
21
22
23  response =
24    Controllers.SavePreferenceController.save_preferences(customer_id,
25                                                         valid_request)
26
27  send_resp(conn, 201, Poison.encode!(response))
28
29  end
30 end
31 end

```

```

1  defmodule Plugs.SaveCustomer do
2    import Plug.Conn
3
4    def init(opts) do
5      opts
6    end
7
8    def call(conn, _opts) do
9      body = conn.body_params
10
11      with {:ok, customer_id} <-
12        Validation.RequestParameterValidation.has_valid_parameters(conn.path_params),
13        :ok <-
14      else
15        {:error, :invalid_url_params} ->
16          send_resp(conn, 400, Poison.encode!(%{error:
17            "URL Parameters need to be alpha numeric"}))
18        {:error, :no_preferences} ->
19          send_resp(conn, 400, Poison.encode!(%{error:
20            "Preferences section missing"}))
21        {:error, :no_accommodation} ->
22          send_resp(conn, 400, Poison.encode!(%{error:
23            "Accommodation section missing"}))
24      end
25    end
26  end
27 end

```

Handle Errors

```

1  defmodule Plugs.SaveCustomer do
2    import Plug.Conn
3
4    def init(opts) do
5      opts
6    end
7
8    def call(conn, _opts) do
9      body = conn.body_params
10
11      with {:ok, customer_id} <-
12         Validation.ParametersValidation.has_valid_parameters(conn.path_params),
13
14         Validation.PreferencesValidation.has_preferences(body),
15
16         :ok <-
17         Validation.AccommodationValidation.has_accommodation_preferences(body) do
18
19         valid_request = Schema.Http.Request.new!(body, customer_id, body["version"])
20         response =
21           Controllers.SavePreferenceController.save_preferences(customer_id, valid_request)
22         send_resp(conn, 201, Poison.encode!(response))
23       else
24         {:error, :invalid_url_params} ->
25           send_resp(conn, 400, Poison.encode!(%{error: "URL Parameters need to be alpha numeric"}))
26         {:error, :no_preferences} ->
27           send_resp(conn, 400, Poison.encode!(%{error: "Preferences section missing"}))
28         {:error, :no_accommodation} ->
29           send_resp(conn, 400, Poison.encode!(%{error: "Accommodation section missing"}))
30       end
31     end
32   end
33 end

```

Add new modules which checks the new rule


```

1  defmodule Plugs.SaveCustomer do
2    import Plug.Conn
3
4    def init(opts) do
5      opts
6    end
7
8    def call(conn, _opts) do
9      body = conn.body_params
10
11      with {:ok, customer_id} <-
12         Validation.ParametersValidation.has_valid_parameters(conn.path_params),
13
14         Validation.PreferencesValidation.has_preferences(body),
15
16         :ok <-
17         Validation.AccommodationValidation.has_accommodation_preferences(body) do
18
19         valid_request = Schema.Http.Request.new!(body, customer_id, body["version"])
20         response =
21           Controllers.SavePreferenceController.save_preferences(customer_id, valid_request)
22         send_resp(conn, 201, Poison.encode!(response))
23       else
24         {:error, :invalid_url_params} ->
25           send_resp(conn, 400, Poison.encode!(%{error: "URL Parameters need to be alpha numeric"}))
26         {:error, :no_preferences} ->
27           send_resp(conn, 400, Poison.encode!(%{error: "Preferences section missing"}))
28         {:error, :no_accommodation} ->
29           send_resp(conn, 400, Poison.encode!(%{error: "Accommodation section missing"}))
30       end
31     end
32   end
33 end

```

Add new modules which checks the new rule

Handle error case



©2013 Bar Products.com

- Don't want to open up the Plug module each time a new validation rule is added
- Don't want to have to test all the rule permutations at this high level



- Do want the ability to plug in new validation rules
- Do want to configure one rule for test env
- Do want to configure all rules for dev/prod env



Categorise Validations

```
with {:ok, customer_id} <-  
  Validation.RequestParameterValidation  
    .has_valid_parameters(conn.path_params),  
  :ok <-  
    Validation.PreferencesValidation  
      .has_preferences(body),  
  :ok <-  
    Validation.AccommodationValidation  
      .has_accommodation_preferences(body) do  
  ...  
end
```

Categorise Validations

```
with {:ok, customer_id} =  
  Validation.ParameterValidation  
    .has_valid_parameters(conn.path_params),  
  :ok <-  
    Validation.PreferencesValidation  
      .has_preferences(body),  
  :ok <-  
    Validation.AccommodationValidation  
      .has_accommodation_preferences(body) do  
  ...  
end
```

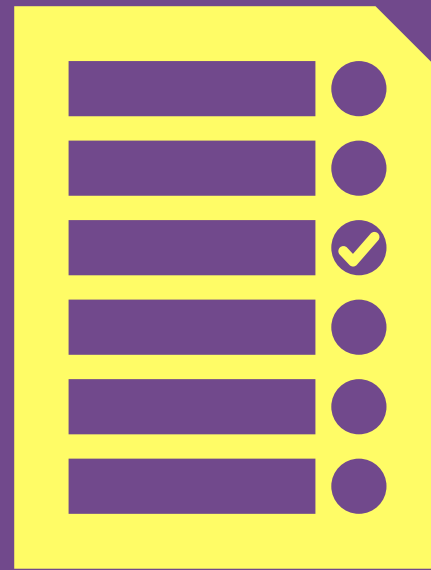
Header

Categorise Validations

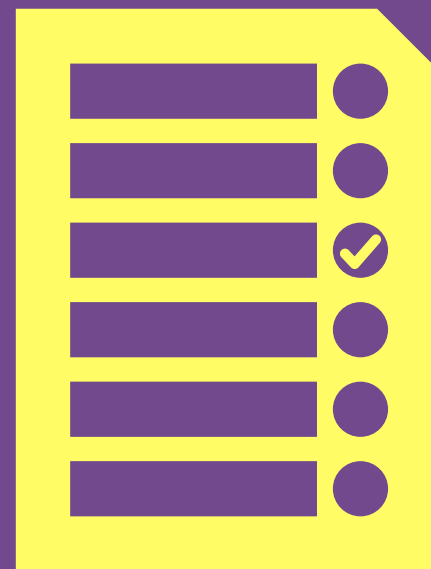
```
with {:ok, customer_id} <-  
  Validation.Header.ParameterValidation  
    .has_valid_parameters(conn.path_params),  
  :ok <-  
    Validation.Body.PreferencesValidation  
      .has_preferences(body),  
  :ok <-  
    Validation.Body.AccommodationValidation  
      .has_accommodation_preferences(body) do
```

...

Define Validation Rules



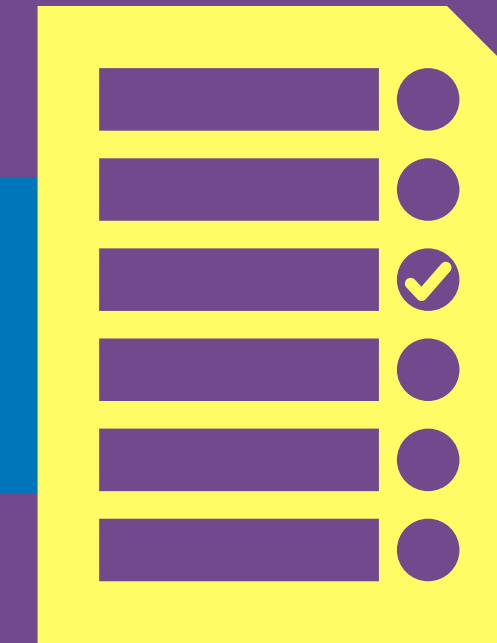
[HeaderRule1, HeaderRule2,
HeaderRule3...]



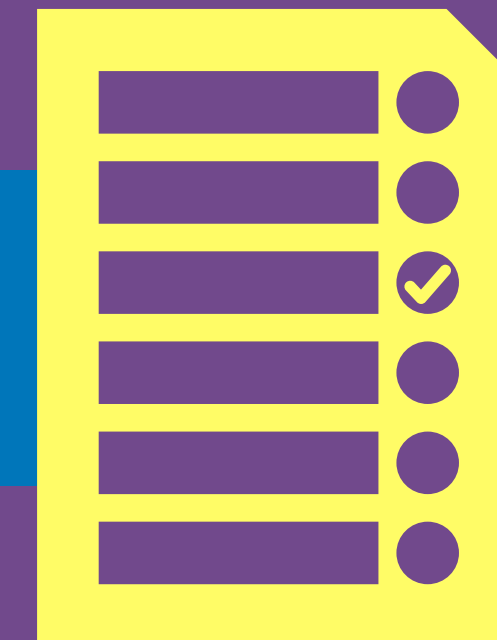
[BodyRule1, BodyRule2, BodyRule3...]

Categorise Validations

```
def module Validation.HeaderValidator
```



```
def module Validation.BodyValidator
```




```

1 ▾ def module Validation.HeaderValidator do
2 ▾   def validate(params) do
3     parameter_validators =
4       [
5         &Validation.RequestParameterValidator.
6           has_valid_parameters/1,
7         &Validation.CustomerRestriction.unrestricted/1]
8
9
10    def validate(params, header_validators) do
11      validate_all_rules(header_validators, params)
12    end
13
14    defp validate_all_rules([], params) do
15      {:ok, params["id"]}
16    end
17 ▾  defp validate_all_rules([rule|tail], params) do
18    {status, data} = rule.(params)
19
20    if status == :error do
21      {:error, data}
22    else
23      validate_all_rules(tail, params)
24    end
25  end
26 end

```

Configure Rules

```
1 ▾ defmodule Validation.HeaderValidator do
2 ▾   def validate(params) do
3 ▾     parameter_validators =
4       [&Validation.RequestParameterValidator.has_valid_parameters/1,
5        &Validation.CustomerRestriction.unrestricted/1]
6
7     validate(params, parameter_validators)
8   end
9
10  def validate(params, header_validators) do
11    validate_all_rules(header_validators, params)
12  end
13
14  defp validate_all_rules([], params) do
15    {:ok, params["id"]}
16  end
```

```
17 ▾ defp validate_all_rules([rule|tail], params) do
18   {status, data} = rule.(params)
19
20   if status == :error do
21     {:error, data}
22   else
23     validate_all_rules(tail, params)
24   end
25 end
```

Traverse and Execute Rules

Save Customer

```
1 defmodule Plugs.SaveCustomer do
2   import Plug.Conn
3
4   def init(opts) do
5     opts
6   end
7
8   def call(conn, _opts) do
9     path_params = conn.path_params
10    body = conn.body_params
11
12    with {:ok, customer_id} <-
13         Validation.HeaderValidator.validate(path_params),
14         {:ok, valid_body} <-
15             Validation.BodyValidator.validate(body) do
16
17
18      send_resp(conn, 201, Poison.encode!(response))
19    else
20      {:error, {_, reason}} -> send_resp(conn, 400, Poison.encode!(%{error: reason}))
21    end
22  end
23 end
```

Validation Per Category

Save Customer

```
1 defmodule Plugs.SaveCustomer do
2   import Plug.Conn
3
4   def init(opts) do
5     opts
6   end
7
8   def call(conn, _opts) do
9     path_params = conn.path_params
10    body = conn.body_params
11
12    with {:ok, customer_id} <-
13         Validation.HeaderValidator.validate(path_params),
14         {:ok, valid_body} <-
15             Validation.BodyValidator.validate(body) do
16
17
18      send_resp(conn, 201, Poison.encode!(response))
19    else
20      {:error, {_, reason}} -> send_resp(conn, 400, Poison.encode!(%{error: reason}))
21    end
22  end
23 end
```





```
1 ▾ defmodule Validation.HeaderValidator do
2 ▾   def validate(params) do
3 ▾     parameter_validators =
4     [&Validation.RequestParameterValidator.
5     has_valid_parameters/1,
6     &Validation.CustomerRestriction.unrestricted/1]
7
8
9
10  def validate(params, header_validators) do
11    validate_all_rules(header_validators, params)
12  end
13
14  defp validate_all_rules([], params) do
15    {:ok, params["id"]}
16  end
17 ▾ defp validate_all_rules([rule|tail], params) do
18    {status, data} = rule.(params)
19
20    if status == :error do
21      {:error, data}
22    else
23      validate_all_rules(tail, params)
24    end
25  end
26 end
```



Behaviours

Behaviours

- “Behaviours provide a way to:
- define a set of functions that have to be implemented by a module
 - ensure that a module implements all of the functions in that set.”

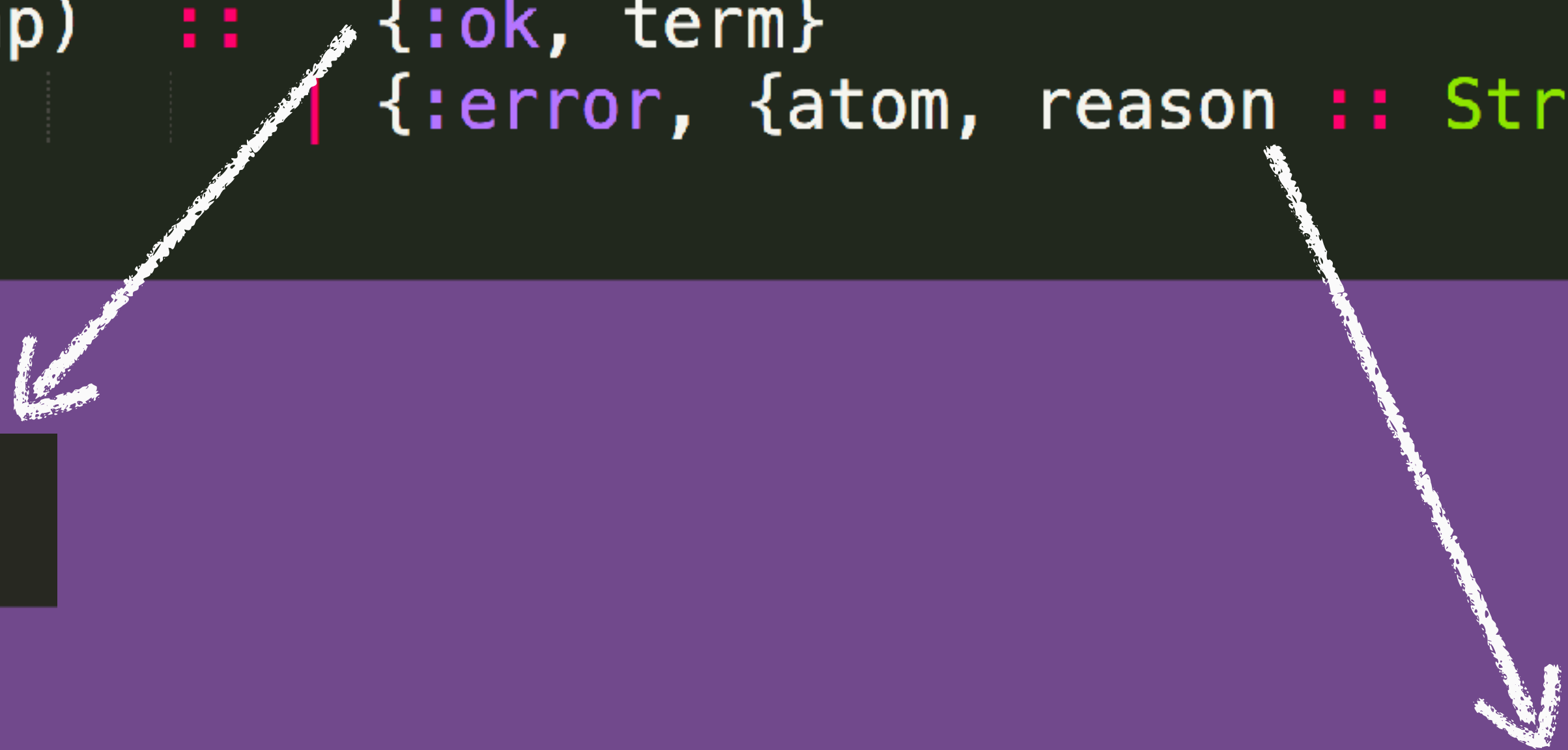
<https://elixir-lang.org/getting-started/typespecs-and-behaviours.html#behaviours>

Validation Rule Behaviour

```
defmodule Validation do
  @callback is_valid(map) :: { :ok, term }
                                | { :error, {atom, reason :: String.t} }
end
```


Validation Rule Behaviour

```
defmodule Validation do
  @callback is_valid(map) :: {:ok, term}
  | {:error, {atom, reason :: String.t}}
end
```



`{:ok, body}`

`{:error, {:no_accommodation, "Accommodation preferences missing"}}`

Implement the Behaviour

Implement Behaviour

```
1 defmodule Validation.Body.ValidationRule do
2   @behaviour Validation 
3
4   def is_valid(request_body) do
5     ...
6   end
7 end
```


Implement Behaviour

```
1 defmodule Validation.Body.ValidationRule do
2   @behaviour Validation
3
4   def is_valid(request_body) do
5     ...
6   end
7 end
```



```
parameter_validators =  
    [&Validation.RequestParameterValidator.  
                                     has_valid_parameters/1,  
     &Validation.CustomerRestriction.unrestricted/1]
```



```
parameter_validators =  
    [Validation.RequestParameterValidator,  
     Validation.CustomerRestriction]
```

Modules implement Behaviour

```
defp validate_all_rules([rule|tail], params) do
  {status, data} = rule.(params)

  if status == :error do
    {:error, data}
  else
    validate_all_rules(tail, params)
  end
end
```



```
defp validate_all_rules([rule|tail], params) do
  {status, data} = rule.is_valid(params)

  if status == :error do
    {:error, data}
  else
    validate_all_rules(tail, params)
  end
end
```

Execute Contract

Configure the Rules

```
1 ▽ config :customer_preference_api,  
2   header_validators: [Validation.RequestParameterValidator,  
3                       Validation.CustomerRestriction  
4   ],  
5   body_validators: [Validation.Body.PreferencesValidation,  
6                   Validation.Body.AccommodationValidation]  
7
```

Configure the Rules

```
1  defmodule Plugs.SaveCustomer do
2    ...
3
4    def call(conn, _opts) do
5      ...
6
7      header_validators =
8        Application.get_env(:customer_preference_api, :header_validators)
9
10     body_validators =
11       Application.get_env(:customer_preference_api, :body_validators)
12
13
14
15       valid_body = Schema.Http.Request.new!(body, customer_id, body["version"])
16       response = Controllers.SavePreferenceController.save_preferences(customer_id, valid_body)
17       send_resp(conn, 201, Poison.encode!(response))
18     else
19       {:error, {_, reason}} -> send_resp(conn, 400, Poison.encode!(%{error: reason}))
20     end
21   end
22 end
```

To Add a New Rule

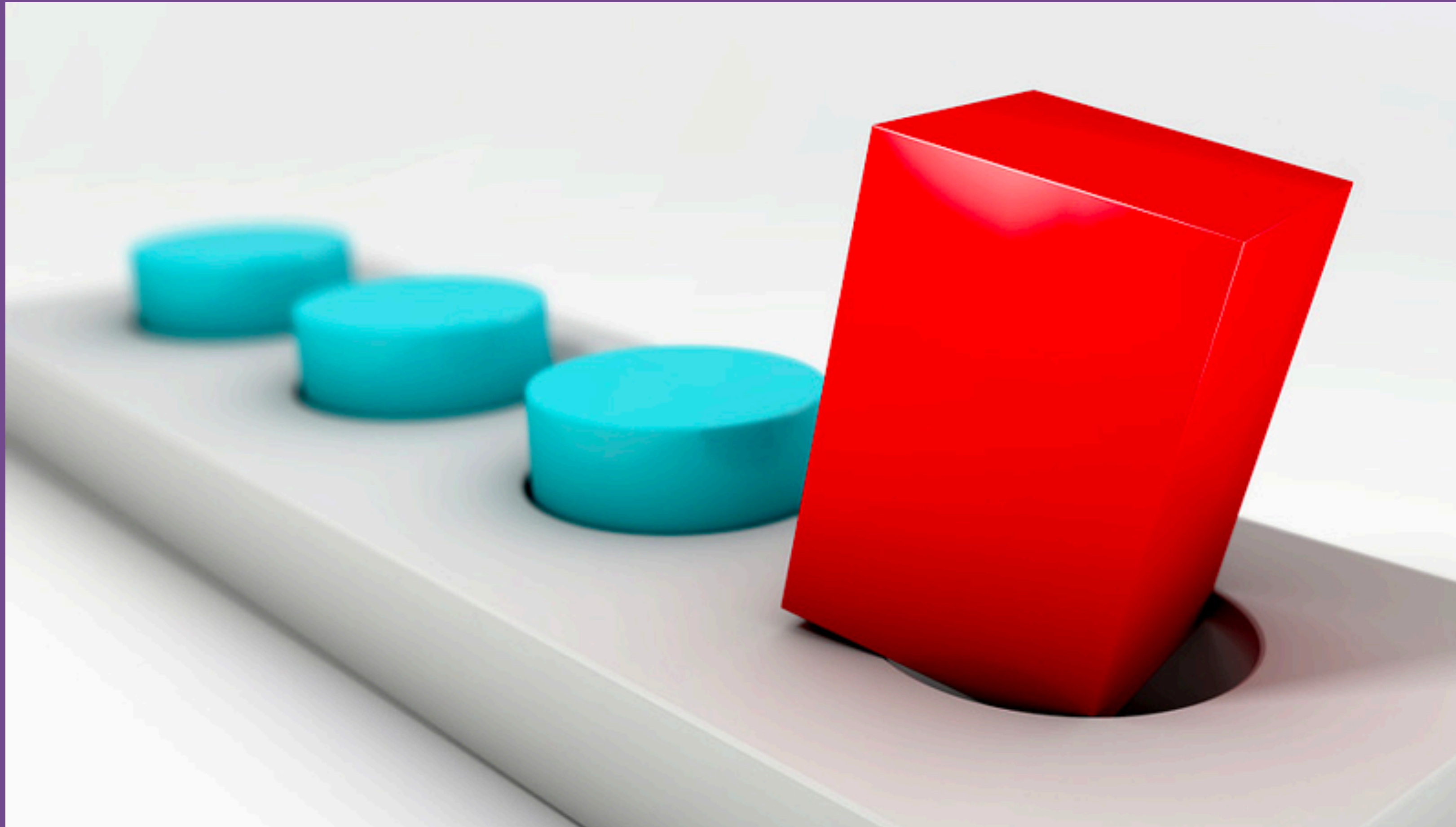
- ✓ Implement New module
 - implements behaviour
 - provides business logic for rule
- ✓ Update the config for the relevant MIX_ENV

You can add rules without
touching any existing code

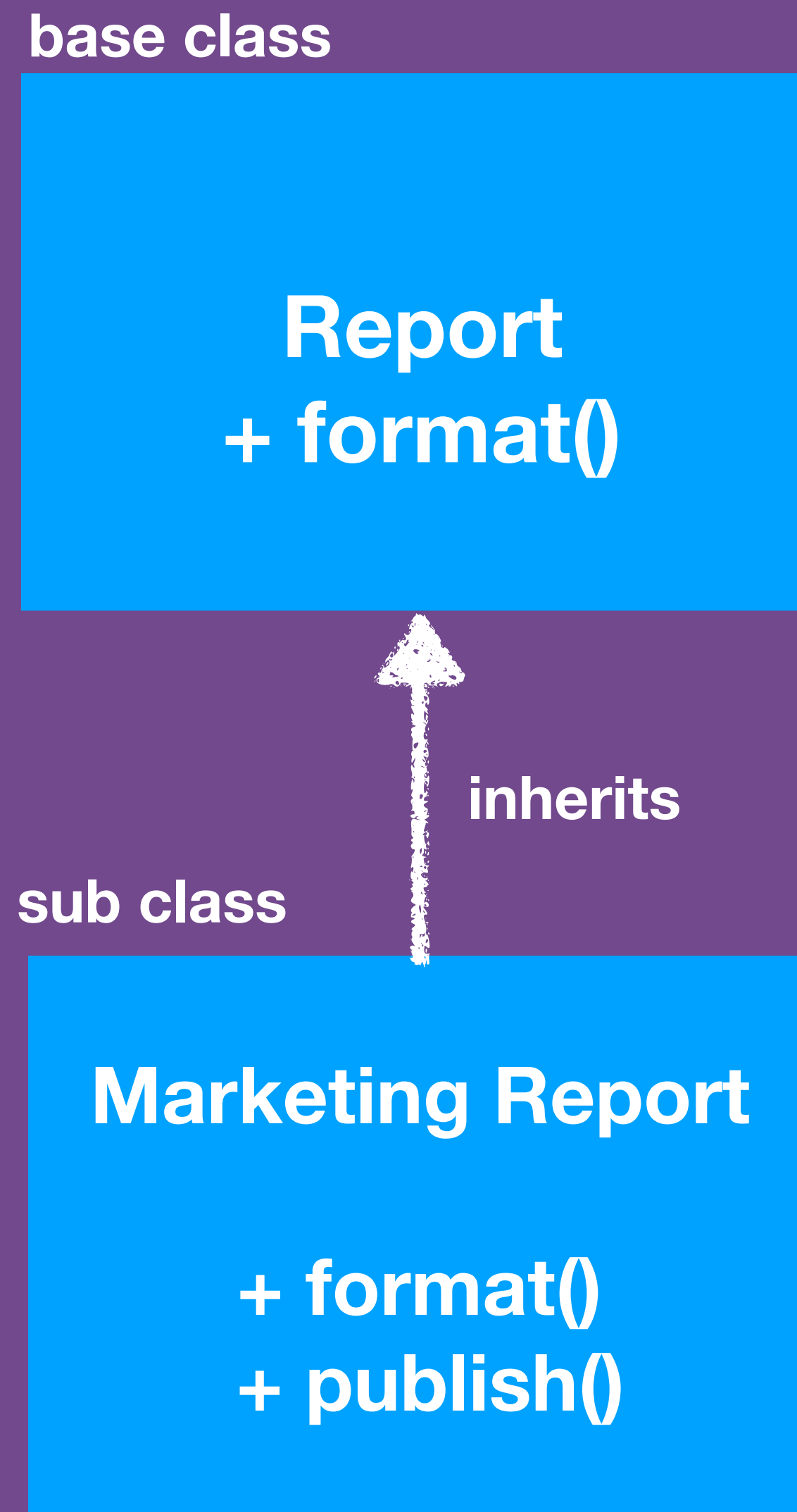


Liskov Substitution

SOLID



Inheritance



“Liskov's notion of a behavioral subtype defines a notion of **substitutability** for objects; that is, if S is a subtype of T , then objects of type T in a program may be **replaced** with objects of type S without altering any of the desirable properties of that program.”

– Wikipedia

https://en.wikipedia.org/wiki/Liskov_substitution_principle

OO Substitution

```
Report marketingReport = new MarketingReport();  
marketingReport.format();
```


OO Substitution

Report



```
Report marketingReport = new MarketingReport();  
marketingReport.format();
```

+ format()

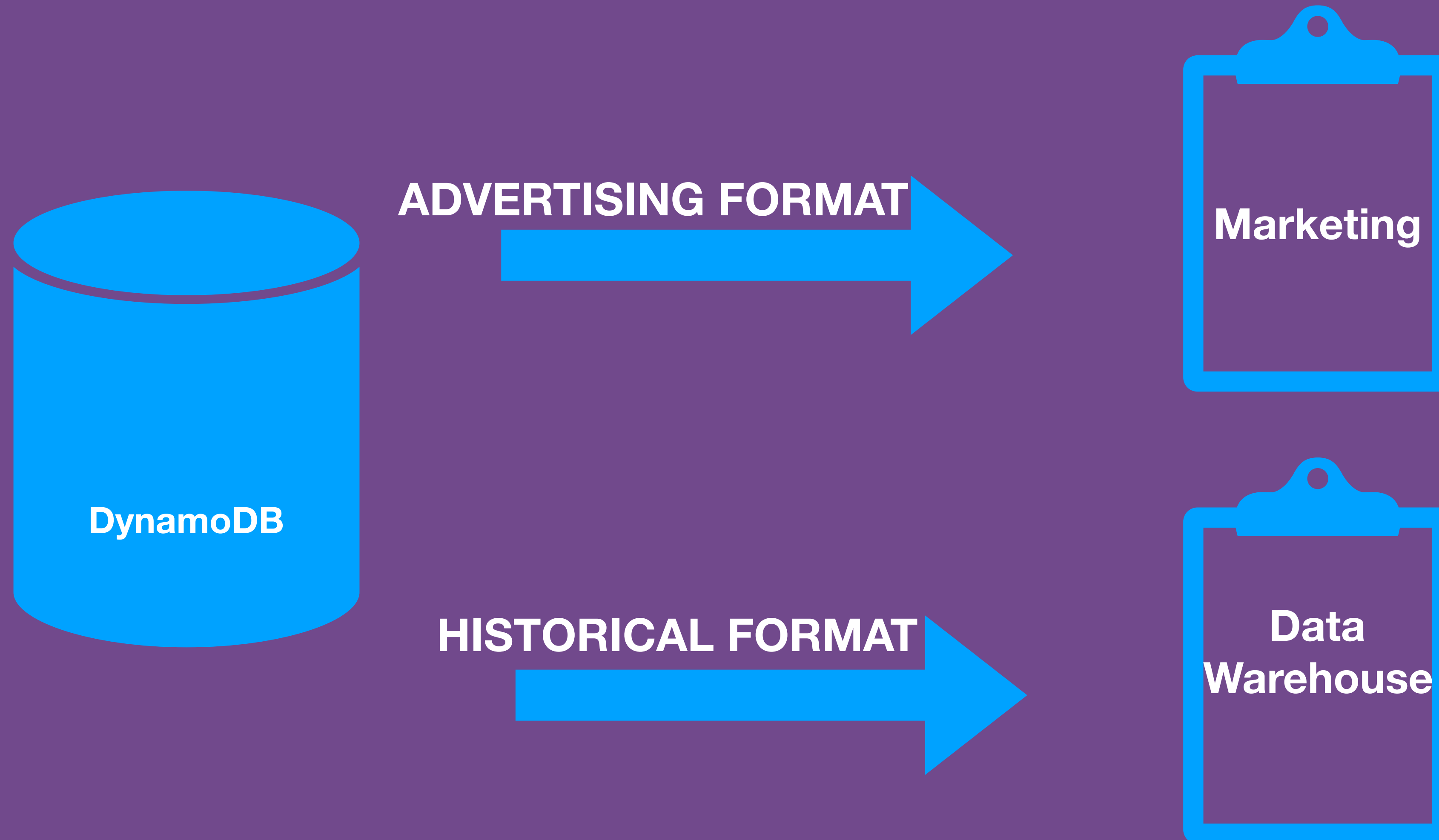
Marketing Report

```
MarketingReport marketingReport = new MarketingReport();  
marketingReport.format();
```

+ format()

+ publish()

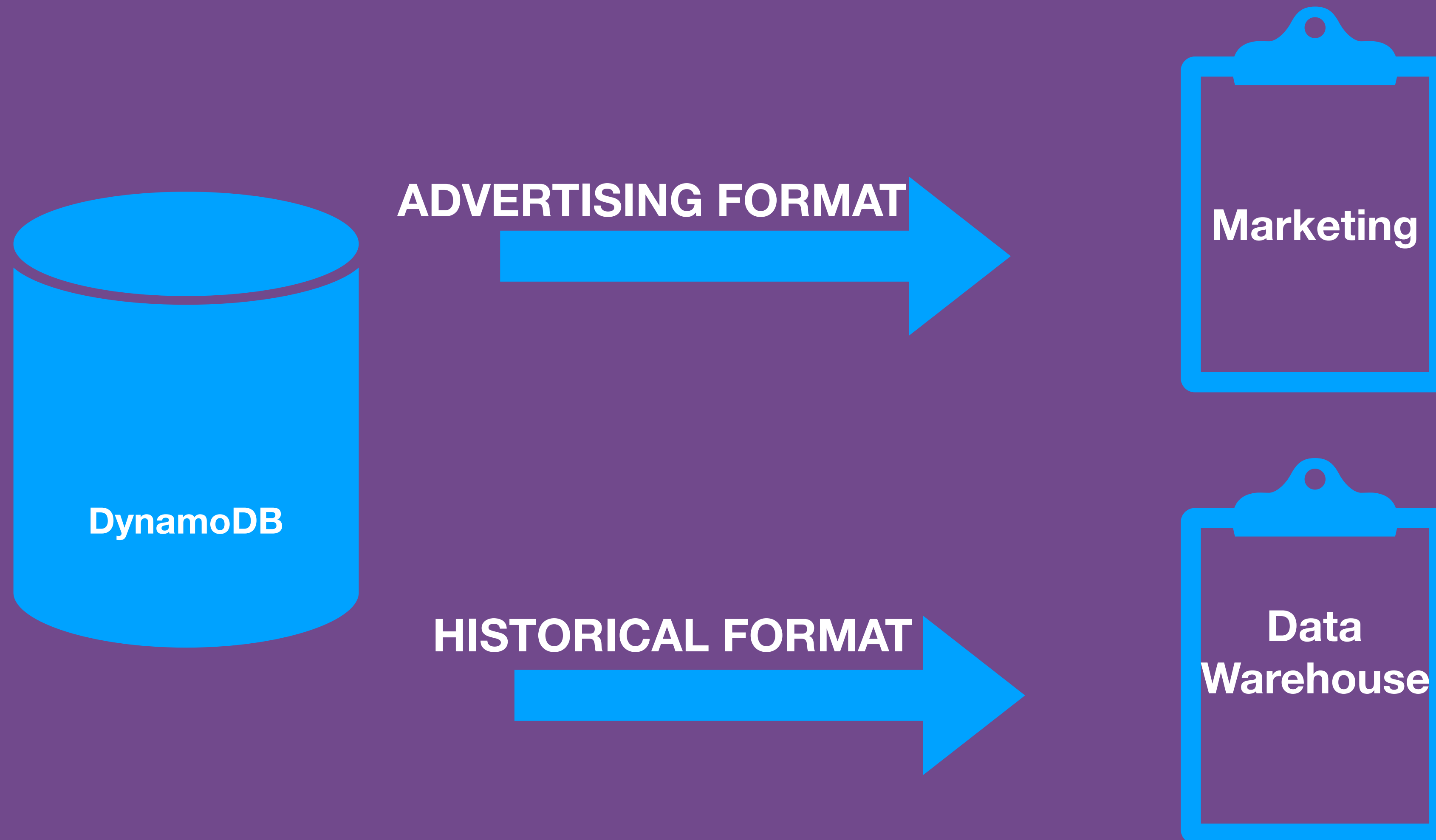
Reporting Capabilities



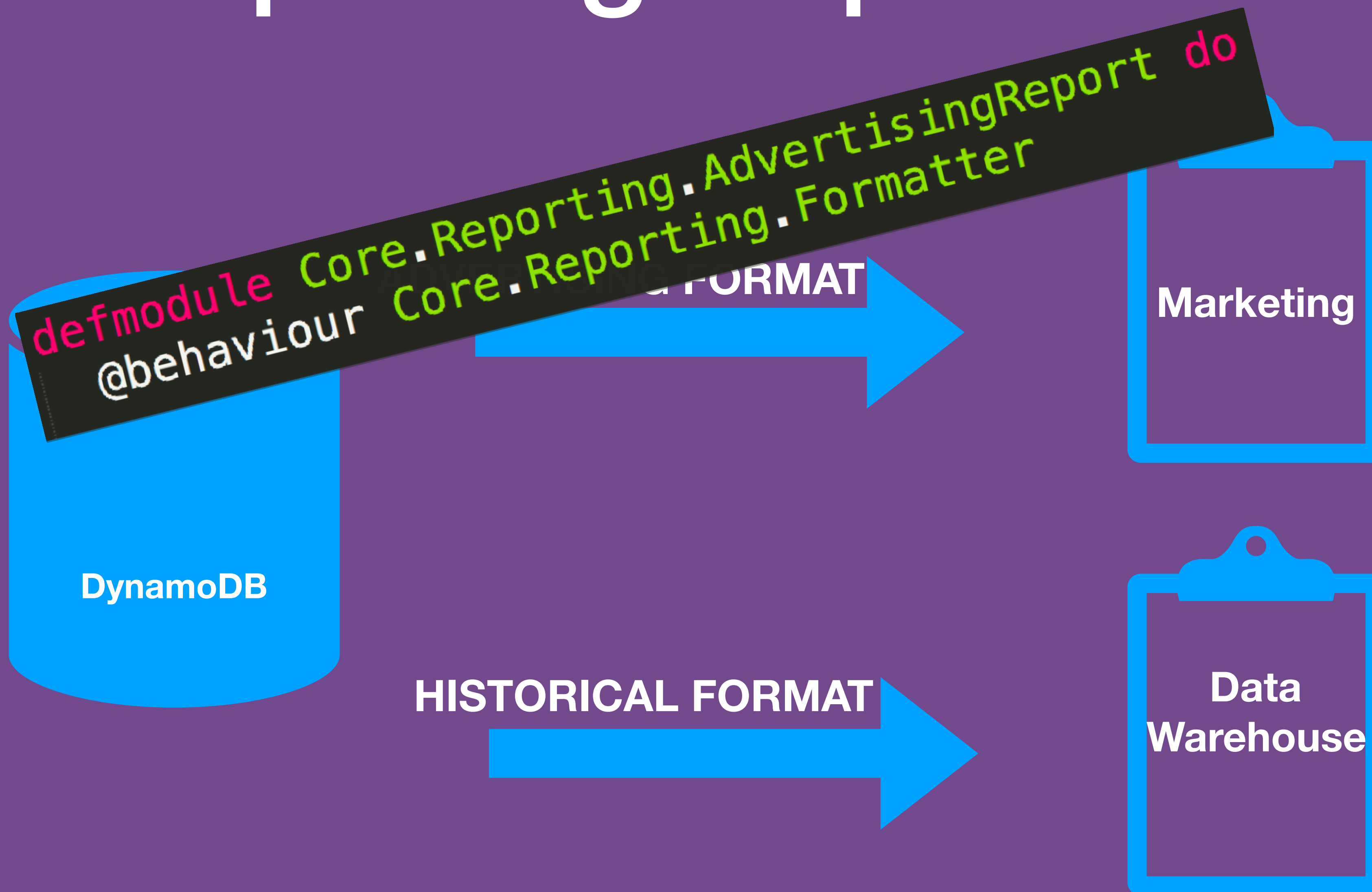
Format Behaviour

```
defmodule Core.Reporting.Formatter do
  @callback format_to_rows(map) :: {:ok, map}
                                     | {:error, String.t}
end
```


Reporting Capabilities

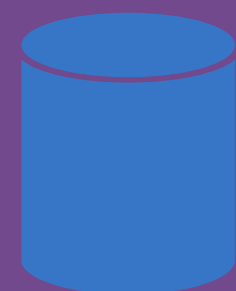


Reporting Capabilities



Reports

```
1 defmodule Core.Reporting.AdvertisingReport do
2   @behaviour Core.Reporting.Formatter 🙋
3
4   @impl true
5   def format_to_rows(data) do 🙋
6     formatted_data = data
7     IO.inspect("format for advertising")
8     {:ok, formatted_data}
9   end
10 end
```



FORMAT



Reporting Capabilities

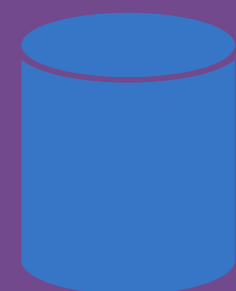


Reporting Capabilities

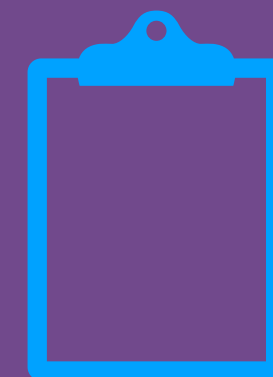


Generation

```
1 defmodule Core.Reporting.ReportGenerator do
2
3   @spec generate_report(data :: map, formats :: list) :: atom
4   def generate_report(data, formats) do
5     formatted_reports = Enum.map(formats, fn(format) ->
6       format.format_to_rows(data)
7     end)
8     dispatch(formatted_reports)
9   end
10
11   def dispatch(reports) do
12     IO.inspect("send to external systems")
13     ...
14     :ok
15   end
16 end
```



FORMAT



Generation

```
1 defmodule Core.Reporting.ReportGenerator do
2
3   @spec generate_report(data :: map, formats :: list) :: atom
4   def generate_report(data, formats) do
5
6     formatted_reports = Enum.map(formats, fn(format) ->
7       format.format_to_rows(data)
8     end)
9
10
11   def dispatch(reports) do
12     IO.inspect("send to external systems")
13     ...
14     :ok
15   end
16 end
```

Liskov Substitution

 Add any new module which implements the Behaviour without altering any desirable properties

Legal Report



Legal Report

- Format as per HistoricalDataReport
- Attach a disclaimer
- Attach a header
- Apply colour to headers

Legal Report

```
1 ▸ defmodule Core.Reporting.LegalReportGenerator do
2
3   def generate_report(data, formatter) do
4     formatter.format_to_rows(data)
5     |> formatter.add_disclaimer
6     |> formatter.add_header
7     |> formatter.colour
8     |> dispatch()
9     :ok
10  end
11
12  def dispatch(report) do
13    IO.inspect("send to external systems")
14    :ok
15  end
16 end
```

```
1 ▾ defmodule Core.Reporting.HistoricalDataReport do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     formatted_data = data
7     IO.inspect("format for historical data")
8     {:ok, formatted_data}
9   end
10
11   def add_disclaimer(data) do
12     IO.inspect("..adding disclaimer..")
13     data
14   end
15
16   def add_header(data) do
17     IO.inspect("..adding header..")
18     data
19   end
20
21   def colour(data) do
22     IO.inspect("..adding colour..")
23     data
24   end
25 end
```



```
1 ▾ defmodule Core.Reporting.HistoricalDataReport do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     formatted_data = data
7     IO.inspect("format for historical data")
8     {:ok, formatted_data}
9   end
10
11   def add_disclaimer(data) do
12     IO.inspect("..adding disclaimer..")
13     data
14   end
15
16   def add_header(data) do
17     IO.inspect("..adding header..")
18     data
19   end
20
21   def colour(data) do
22     IO.inspect("..adding colour..")
23     data
24   end
25 end
```


Generate Legal Report

```
def generate_report(data, formatter) do
  formatter.format_to_rows(data)
  |> formatter.add_disclaimer
  |> formatter.add_header
  |> formatter.colour
  |> dispatch()
  :ok
end
```

Generate Legal Report

```
iex(2)> Core.Reporting.LegalReportGenerator.generate_report(%{"preferences" => "..."},  
  Core.Reporting.HistoricalDataReport)  
"format for historical data"  
"..adding disclaimer.."  
"..adding header.."  
"..adding colour.."  
"send to external systems"  
:ok
```

What happens if you generate
a Legal Report with another
Formatter implementation?

Generate Legal Report

```
def generate_report(data, formatter) do
  formatter.format_to_rows(data)
  |> formatter.add_disclaimer
  |> formatter.add_header
  |> formatter.colour
  |> dispatch()
  :ok
end
```


Generate Legal Report

```
iex(1)> Core.Reporting.LegalReportGenerator.generate_report(%{"preferences" => "..."},
Core.Reporting.AdvertisingReport)
"format for advertising"
** (UndefinedFunctionError) function Core.Reporting.AdvertisingReport.add_disclaimer/1
is undefined or private
    (customer_preference_api) Core.Reporting.AdvertisingReport.add_disclaimer({:ok, %{"preferences" => "..."}})
    (customer_preference_api) lib/core/reporting/legal_report_generator.ex:11: Core.Re
porting.LegalReportGenerator.generate_report/2
iex(1)>
```



Liskov Substitution Violation

 According to Liskov Substitution we should be able to plug any implementation in without altering any of the desirable properties

Liskov Substitution

- ✅ Format as `HistoricalDataReport`
- 🤔 Extra Processing to add headers, colours etc
- 💡 Create a new module that uses the `HistoricalDataReport` and does a bit more


```

1 ▾ def module Core.Reporting.HistoricalDataDecorator do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     IO.inspect("format for historical data")
7     formatted_data =
8     Core.Reporting.HistoricalDataReport.format_to_rows(data)
9     |> add_disclaimer()
10    |> add_header()
11    |> colour()
12    {:ok, formatted_data}
13  end
14
15  def add_disclaimer(data) do
16    IO.inspect("..adding disclaimer..")
17    data
18  end
19
20  def add_header(data) do
21    IO.inspect("..adding header..")
22    data
23  end
24
25  def colour(data) do
26    IO.inspect("..adding colour..")
27    data
28  end
29 end

```



```
1 ▾ def module Core.Reporting.HistoricalDataDecorator do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     IO.inspect("format for historical data")
7 ▾   formatted_data =
```

```
Core.Reporting.HistoricalDataReport.format_to_rows(data)
```

```
9     |> add_disclaimer()
10    |> add_header()
11    |> colour()
12    {:ok, formatted_data}
13  end
14
15  def add_disclaimer(data) do
16    IO.inspect("..adding disclaimer..")
17    data
18  end
19
20  def add_header(data) do
21    IO.inspect("..adding header..")
22    data
23  end
24
25  def colour(data) do
26    IO.inspect("..adding colour..")
27    data
28  end
29 end
```

```

1 ▾ defmodule Core.Reporting.HistoricalDataDecorator do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     IO.inspect("format for historical data")
7     formatted_data =
8     Core.Reporting.HistoricalDataReport.format_to_rows(data)
9     |> add_disclaimer()
10    |> add_header()
11    |> colour()
12
13   end
14
15   def add_disclaimer(data) do
16     IO.inspect("..adding disclaimer..")
17     data
18   end
19
20   def add_header(data) do
21     IO.inspect("..adding header..")
22     data
23   end
24
25   def colour(data) do
26     IO.inspect("..adding colour..")
27     data
28   end
29 end

```

```
defmodule Core.Reporting.ReportGenerator do
```

```
defmodule Core.Reporting.AdvertisingReport do  
  @behaviour Core.Reporting.Formatter
```

```
defmodule Core.Reporting.HistoricalDataReport do  
  @behaviour Core.Reporting.Formatter
```

```
defmodule Core.Reporting.HistoricalDataDecorator do  
  @behaviour Core.Reporting.Formatter
```

Marketing

Data
Warehouse

Legal

Liskov Substitution

- ✓ Wrapping an existing Formatter allowed us to extend it's functionality
- ✓ Substitute any implementation safely

Liskov Substitution

Where we have code that expects
a **generic type** (i.e: Behaviour) -
Ensure you are only using function
calls **defined on that Behaviour**

But why didn't we just
enhance the original
Behaviour with the new
functions?



Update Each Implementation

 Update HistoricalDataReport

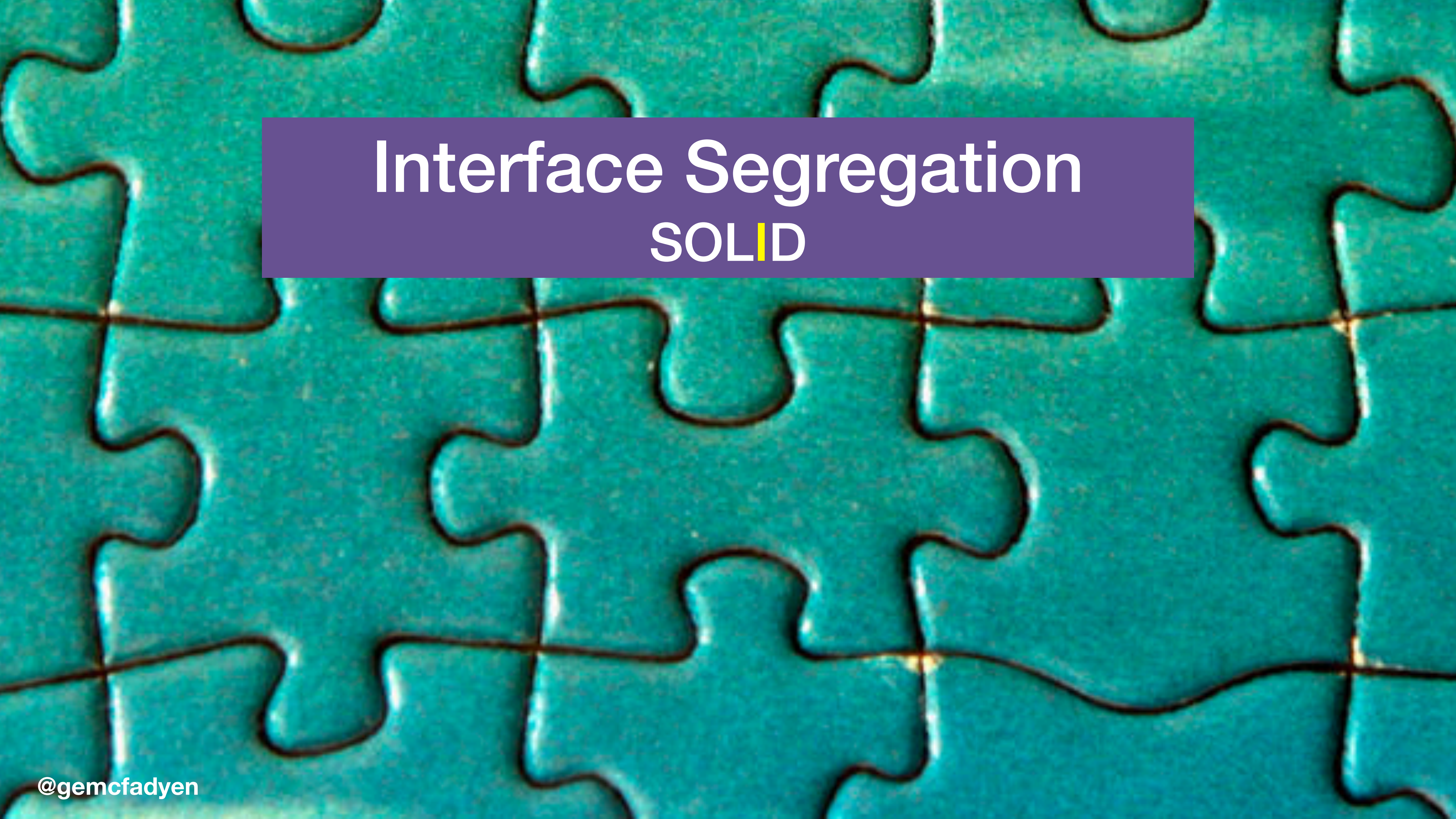
? Update AdvertisingReport


```

1 ▾ defmodule Core.Reporting.AdvertisingReport do
2   @behaviour Core.Reporting.Formatter
3
4   @impl true
5   def format_to_rows(data) do
6     formatted_data = data
7     IO.inspect("format for advertising")
8     #... some data clensing and reformatting logic
9     {:ok, formatted_data}
10  end
11
12  def add_disclaimer(data) do
13    IO.puts("nothing to do..")
14    data
15  end
16
17  def add_header(data) do
18    IO.puts("nothing to do..")
19    data
20  end
21
22  def colour(data) do
23    IO.puts("nothing to do..")
24    data
25  end
26 end

```


 Bloated the Behaviour to satisfy
a new requirement

The background of the slide is a close-up photograph of teal-colored interlocking puzzle pieces. The pieces are arranged in a grid-like pattern, with their characteristic interlocking shapes clearly visible. The lighting creates subtle highlights and shadows on the raised and recessed surfaces of the puzzle pieces, giving the background a textured, three-dimensional appearance.

Interface Segregation SOLID

“The interface-segregation principle (ISP) states that no client should be **forced to depend on methods it does not use.**^[1] ISP splits interfaces that are very large into smaller and more specific ones so that clients will only have to know about the methods that are of interest to them.”

– *Wikipedia*

https://en.wikipedia.org/wiki/Interface_segregation_principle

Interface Segregation Signs

- Look for Behaviours which have methods that don't apply to all implementors
- Look for Behaviours that have many function definitions

Interface Segregation Signs

```
1 defmodule Core.Reporting.Formatter do
2   @callback format_to_rows([map]) :: {:ok, [map]}
3   |                                     | {:error, String.t}
4
5   @callback add_disclaimer([map]) :: [map]
6
7   @callback add_header([map]) :: [map]
8
9   @callback colour([map]) :: [map]
10 end
```

Split Behaviour

```
1 defmodule Core.Reporting.Formatter do
2   @callback format_to_rows([map]) :: {:ok, [map]}
3   | {:error, String.t}
4 end
```

Split Behaviour

```
1 defmodule Core.Reporting.Formatter do
2   @callback format_to_rows([map]) :: {:ok, [map]}
3   |                                     | {:error, String.t}
4 end
```

```
1 ▸ defmodule Reporting.Presentation do
2   @callback add_disclaimer([map]) :: [map]
3
4   @callback add_header([map]) :: [map]
5
6   @callback colour([map]) :: [map]
7 end
```



```

1 defmodule Core.Reporting.HistoricalDataReport do
2   @behaviour Core.Reporting.Formatter
3   @behaviour Core.Reporting.Presentation
4
5   def format_to_rows(data) do
6     formatted_data = data
7     IO.inspect("format for historical data")
8     #... some data cleansing and reformatting logic
9     {:ok, formatted_data}
10  end
11
12  def add_disclaimer(data) do
13    IO.inspect("..adding disclaimer..")
14    #...adds legal disclaimer
15    data
16  end
17
18  def add_header(data) do
19    IO.inspect("..adding header..")
20    data
21  end
22
23  def colour(data) do
24    IO.inspect("..adding colour..")
25    data
26  end
27 end

```



```
defmodule Core.Reporting.ReportGenerator do
  def generate_report(data, formats) do
    formatted_reports = Enum.map(formats, fn(format) ->
      format.format_to_rows(data)
    end)
    dispatch(formatted_reports)
  end
end
```

```
  def present_report(data, presenter_formats) do
    formatted_reports = Enum.map(presenter_formats, fn(presenter_format) ->
      presenter_format.format_to_rows(data)
      |> presenter_format.add_disclaimer()
      |> presenter_format.add_header()
      |> presenter_format.colour()
      |> dispatch
    end)
  end

  def dispatch(reports) do
    IO.inspect("send to external systems")
    :ok
  end
end
```

```
defmodule Core.Reporting.ReportGenerator do
  @spec generate_report(data :: map, formats :: list) :: atom
  def generate_report(data, formats) do
    formatted_reports = Enum.map(formats, fn(format) ->
      format.format_to_rows(data)
    end)
    dispatch(formatted_reports)
  end
end
```

```
@spec present_report(data :: map, presenter_formats:: list) :: atom
def present_report(data, presenter_formats) do
  formatted_reports = Enum.map(presenter_formats, fn(presenter_format) ->
    presenter_format.format_to_rows(data)
    |> presenter_format.add_disclaimer()
    |> presenter_format.add_header()
    |> presenter_format.colour()
    |> dispatch
  end)
end
```

```
def dispatch(reports) do
  IO.inspect("send to external systems")
  :ok
end
end
```

Split Processing

```
1 defmodule Core.Reporting.ReportingClient do
2   def entry_point(data) do
3     formatted_data = generate_report(data,
4                                   [Core.Reporting.TrendsReport,
5                                   Core.Reporting.AdvertisingReport])
6     |> dispatch
7
8     present_report(data,
9                   [Core.Reporting.HistoricalDataReport])
10    |> dispatch
11  end
12 end
```




Force implementations only
on those modules that actually
need them



Dependency Inversion

SOLID

The dependency inversion principle refers to a specific form of **decoupling** software modules.

High-level modules should not depend on low-level modules. Both should **depend on abstractions**.

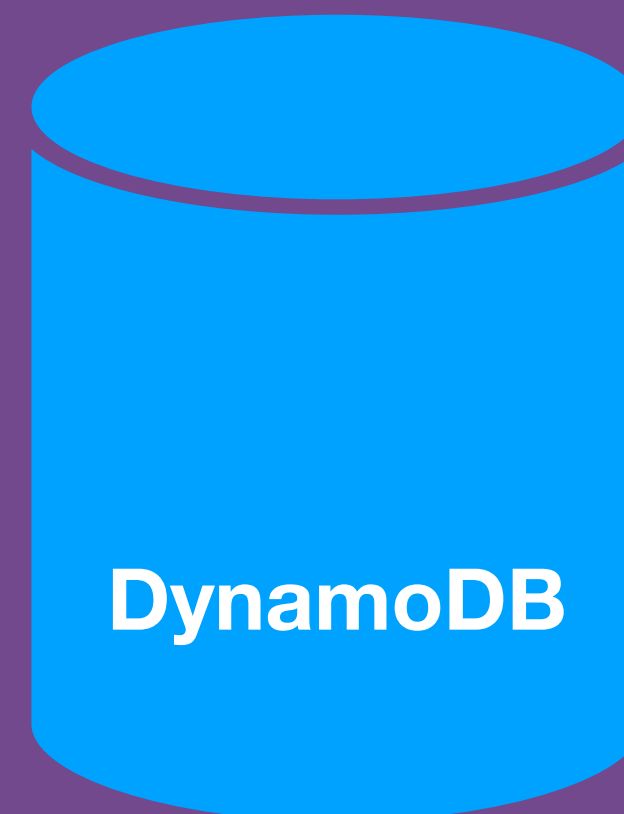
Abstractions should not depend on details.
Details should **depend on abstractions**.

– *Wikipedia*

https://en.wikipedia.org/wiki/Dependency_inversion_principle

GET /customers/{id}

↓ Lookup




```
1 ▾ defmodule Core.Preferences.GetCustomerPreferences do
2
3 ▾   def for(customer_id) do
```

```
ExAws.Dynamo.get_item("customer-preferences",
                        %{id: customer_id})
```

```
|> ExAws.request
```

Fetch Customer Preferences

```
9
10 defp decode({:ok, response}) when map_size(response) == 0 do
11   {:error, :not_found}
12 end
13 ▾ defp decode({:ok, response}) do
14 ▾   valid_consent_user =
15     ExAws.Dynamo.decode_item(response,
16       as: Schema.Database.CustomerPreferencesRow)
17
18   {:ok, valid_consent_user}
19 end
20 defp decode({:error, response}) do
21   {:error, response}
22 end
23 end
```

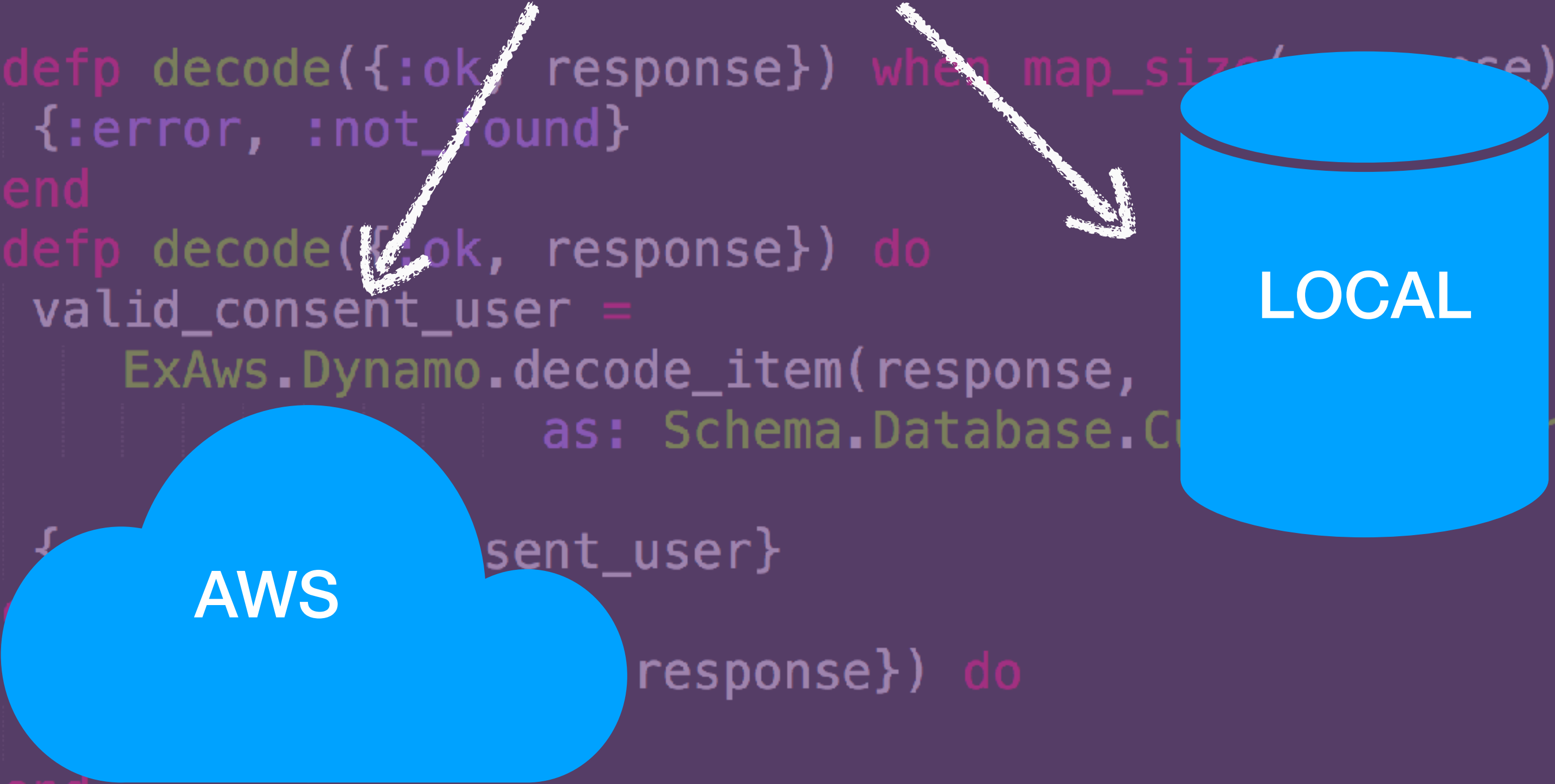


```
1 ▾ defmodule Core.Preferences.GetCustomerPreferences do
2
3 ▾   def for(customer_id) do
```

```
ExAws.Dynamo.get_item("customer-preferences",
                        %{id: customer_id})
```

```
|> ExAws.request
```

```
9
10 defp decode({:ok, response}) when map_size(response) == 0 do
11   {:error, :not_found}
12 end
13 ▾ defp decode({:ok, response}) do
14 ▾   valid_consent_user =
15     ExAws.Dynamo.decode_item(response,
16                             as: Schema.Database.ConsentUserRow)
17
18   {valid_consent_user}
19
20   decode(response) do
21
22   end
23 end
```



```
1 ▾ defmodule Core.Preferences.GetCustomerPreferences do
2
3 ▾   def for(customer_id) do
```

```
ExAws.Dynamo.get_item("customer-preferences",
                        %{id: customer_id})
```

```
|> ExAws.request
```

```
9
10 defp decode({:ok, response}) when map_size(response) == 0 do
11   {:error, :not_found}
12 end
13 ▾ defp decode({:ok, response}) do
14 ▾   valid consent user =
15     _item(response,
16           schema.Database.CustomerPreferencesRow)
17
18   {
19     en
20     de(response)) do
21     {
22     end
23 end
```

FAKE DATABASE

MIX_ENV = test
- fake-database

MIX_ENV = dev
- local-database

MIX_ENV = prod
- aws-database

Isolation

```
def for(customer_id) do
  ExAws.Dynamo.get_item("customer-preferences",
                        %{id: customer_id})
  |> ExAws.request
  |> decode
end
```



```
|> ExAws.request
```


Isolation

```
def for(customer_id) do
  ExAws.Dynamo.get_item("customer-preferences",
                        %{id: customer_id})
  |> ExAws.request
  |> decode
end
```



ExAws.request

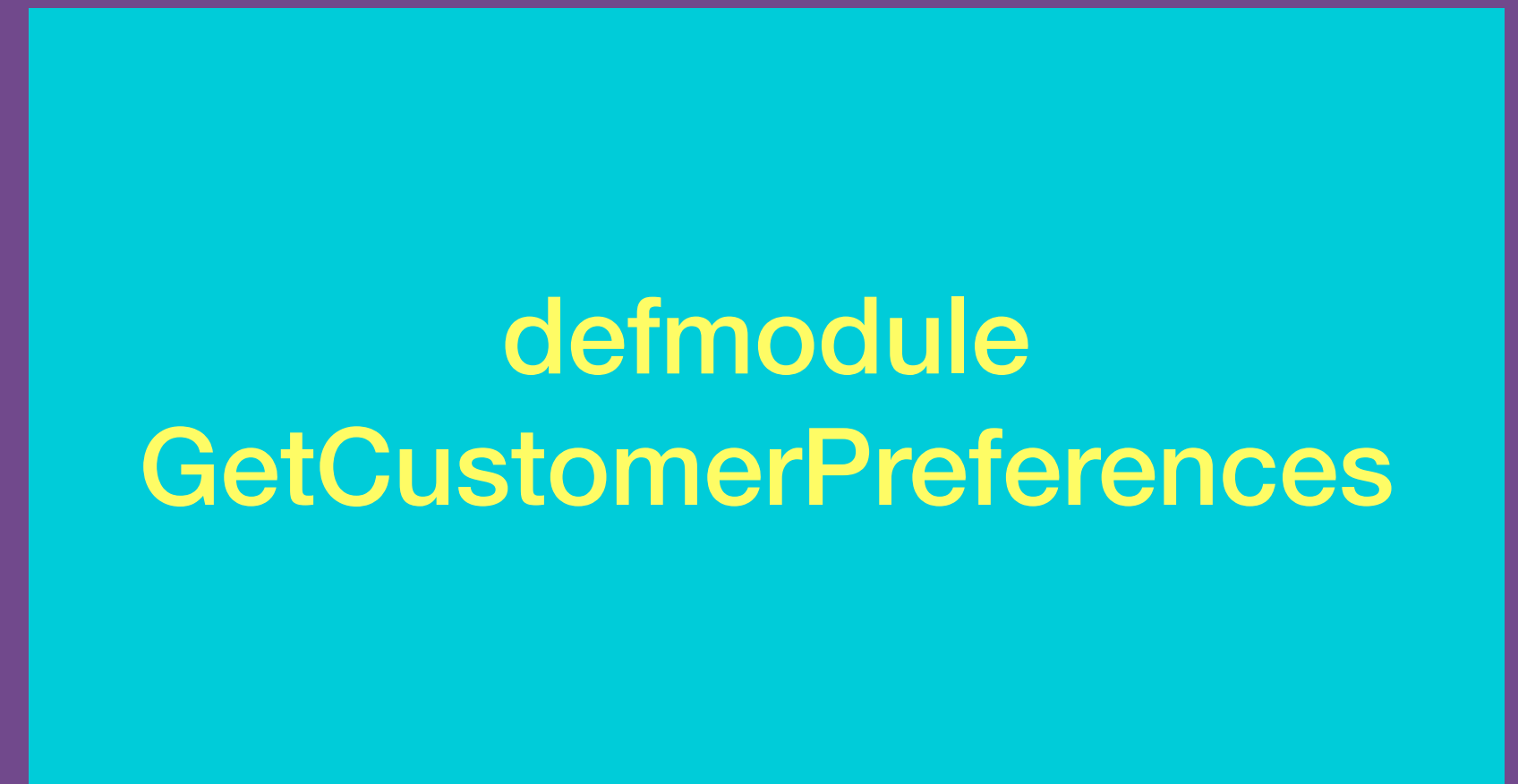
FakeDatabase.request

Use Config to Substitute Test Doubles

defmodule AwsRequest



*abstraction
(Behaviour)*



defmodule FakeDatabase



defmodule DatabaseRequest

Create Abstraction

```
1 ▾ defmodule Database.Request do
2   @callback request(ExAws.Operation.t, Keyword.t) :: {:ok, term}
3   | {:error, term}
4
5   @callback request(ExAws.Operation.t) :: {:ok, term}
6   | {:error, term}
7 end
```


Wrap the real database functionality

```
1 ▾ defmodule Database.AwsRequest do
2   @behaviour Database.Request
3
4   @impl true
5   def request(op, config_overrides \\ []) do
6     ExAws.request(op, config_overrides)
7   end
8 end
```

```
1 ▾ defmodule Database.AwsRequest do
2   @behaviour Database.Request
3
4   @impl true
5   def request(op, config_overrides \\ []) do
6     ExAws.request(op, config_overrides)
7   end
8 end
```

**Provide a fake database
request module**


```

1  defmodule Database.FakeDatabase do
2    @behaviour Database.Request
3
4    @impl true
5    def request(op, config_overrides \\ [])
6
7    @impl true
8    def request(%{http_method: :post}, _) do
9      {:ok, canned_result()}
10   end
11
12   def canned_result do
13     IO.puts("Saving data...")
14
15     %{
16       "Item" =>
17         %{
18           "id" => %{
19             "S" => "uuid-1",
20             "preferences" => %{
21               "M" =>
22                 %{
23                   "accommodation" => %{
24                     "M" =>
25                       %{
26                         "apartment" => %{
27                           "M" =>
28                             %{
29                               "bedrooms" => %{
30                                 "N" => "2",
31                                 "catering" => %{
32                                   "S" => "self_catering",
33                                   "parking" => %{
34                                     "S" => "secure"
35                                   }
36                                 }
37                               }
38                             }
39                           }
40                         }
41                       }
42                     }
43                   }
44                 }
45             }
46           }
47         }
48     }
49   end
50 end

```


MIX_ENV=test

fake_database.ex

MIX_ENV=prod

aws_request.ex

customer_preference_api/config/test.exs

```
config :customer_preference_api,  
  aws_request: Database.FakeDatabase
```

customer_preference_api/config/prod.exs

```
config :customer_preference_api,  
  aws_request: Database.AwsRequest
```

Lookup implementation

```
1 defmodule Core.Preferences.GetCustomerPreferences do
2
3   def for(customer_id) do
4     database = Application.get_env(:customer_preference_api,
5                                   :aws_request)
6
7     ExAws.Dynamo.get_item("customer-preferences", %{id: customer_id})
8     |> database.request
9     |> decode
10  end
11  ...
12 end
```


Configure different Scenarios

- Testing scenarios where nothing is found
- Testing scenarios where multiple entries found....

Configure Fake via Config

- ✓ Isolates 'fake' functionality to specific MIX_ENV
- ✗ Messy to handle different canned results for different scenarios
- ✗ Difficult to know what 'fake' logic is being executed for what test
- ✗ Fake implementations can diverge from the real implementations

Mocking Library

<https://github.com/plataformatec/mox>

Add dependency

```
1  defp deps do
2  ▼  [
3      ...
4      {:mox, "~> 0.3.1"},
5      ...
6  ]
7  end
```

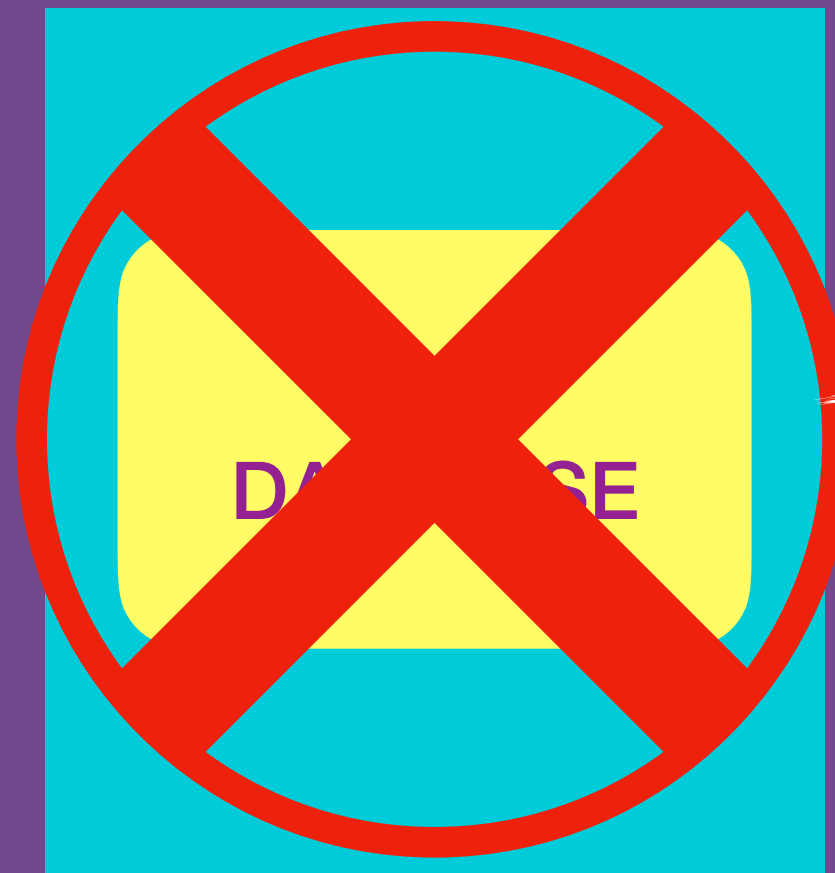

defmodule AwsRequest



*abstraction
(Behaviour)*

**defmodule
GetCustomerPreferences**

defmodule FakeDatabase



defmodule DatabaseRequest

Wrap 'real' implementation

```
1 ▾ defmodule Database.AwsRequest do
2   @behaviour Database.Request
3
4   @impl true
5   def request(op, config_overrides \\ []) do
6     ExAws.request(op, config_overrides)
7   end
8 end
```

Configure Implementations

- prod config

```
1 config :customer_preference_api,  
2   aws_request: Database.AwsRequest
```

Configure Implementations

- test config

```
1 config :customer_preference_api,  
2   aws_request: Database.MockRequest,
```


Test Expectations

```
1 ExUnit.start()  
2  
3 Mox.defmock(Database.MockRequest, for: Database.Request)
```


Configure Implementations

```
1  defmodule Core.Preferences.GetCustomerPreferences do
2    @aws_request Application.get_env(:customer_preference_api,
3                                   :aws_request)
4
5    def for(customer_id) do
6      ExAws.Dynamo.get_item("customer-preferences",
7                            %{id: customer_id})
8      |> @aws_request.request
9      |> decode
10   end
11   ...
12 end
```

Test Expectations

```
1 defmodule Core.Preferences.GetCustomerPreferencesTest do
2   use ExUnit.Case
3
4   import Mox
5   @mock_request Database.MockRequest
6
7   expect(@mock_request, :request, fn _ -> {:ok, canned_result()} end)
8
9   {:ok, result} = Core.Preferences.GetCustomerPreferences.for("1")
10
11   assert result.id == "uuid-1"
12   assert result.preferences["accommodation"]["apartment"]["bedrooms"] == 2
13   assert result.preferences["accommodation"]["hotel"]["catering"] == "catered"
14 end
15
16 def canned_result do
17   ...
18 end
19 end
```

Test Expectations

```
1 defmodule Core.Preferences.GetCustomerPreferencesTest do
2   use ExUnit.Case
3   import Mox
4   @mock_request Database.MockRequest
5
6   test "fetches preferences from database" do
7     expect(@mock_request,
8           :request,
9           fn _ -> {:ok, canned_result()} end)
10
11     result = Core.Preferences.GetCustomerPreferencesTest.canned_result()
12     assert result.preferences["accommodation"]["hotel"]["catering"] == "catered"
13   end
14
15   def canned_result do
16     ...
17   end
18 end
19 end
```

Define expectation

Test Expectations

```
1) test fetches preferences from database (Core.Preferences.GetCustomerPreferencesTest)
   test/core/preferences/get_customer_preferences_test.exs:7
   ** (Mox.UnexpectedCallError) no expectation defined for Database.MockRequests.request/1 in process #PID<0.591.0>
   code: {:ok, result} = Core.Preferences.GetCustomerPreferences.for("1")
   stacktrace:
     (mox) lib/mox.ex:438: Mox.__dispatch__/4
     (customer_preference_api) lib/core/preferences/get_customer_preferences.ex:6: Core.Preferences.GetCustomerPreferences.for/1
     test/core/preferences/get_customer_preferences_test.exs:9: (test)
```


Mocking Libraries

- ✓ Easy to setup different scenarios
- ✓ Fake behaviour lives with the test
- ✓ Don't have to maintain any 'fake' implementations
- ✗ Mix spits out some warnings about the mock modules not physically existing

Dependency Inversion

SOLID

Depend on abstractions

Provide to the application, a specific implementation of a dependency to use depending on the circumstance

Can
you
write
SOLID
Elixir?



Can
you
write
SOLID
Elixir?

yes!



Do we need a set of
functional design
principles?

SOLID 

Functions

SOLID 

Higher Order Functions

SOLID

Pure Functions &
Referential Transparency

Pure Functions

Deterministic with no side effects

input -> output

```
def function(input) do
  output
end
```

Pure Functions

```
def add(number, number) do
  number + number
end
```


Pure Functions

```
def add(number, number) do  
  number + number  
end
```

```
def lucky_number() do  
  add(2, 2)  
  |> add(2)  
  |> add(1)  
end
```


Referential Transparency

```
def add(number, number) do  
  number + number  
end
```

```
def lucky_number() do  
  add(2, 2)  
  |> add(2)  
  |> add(1)  
end
```

7

SOLID 

Behaviour

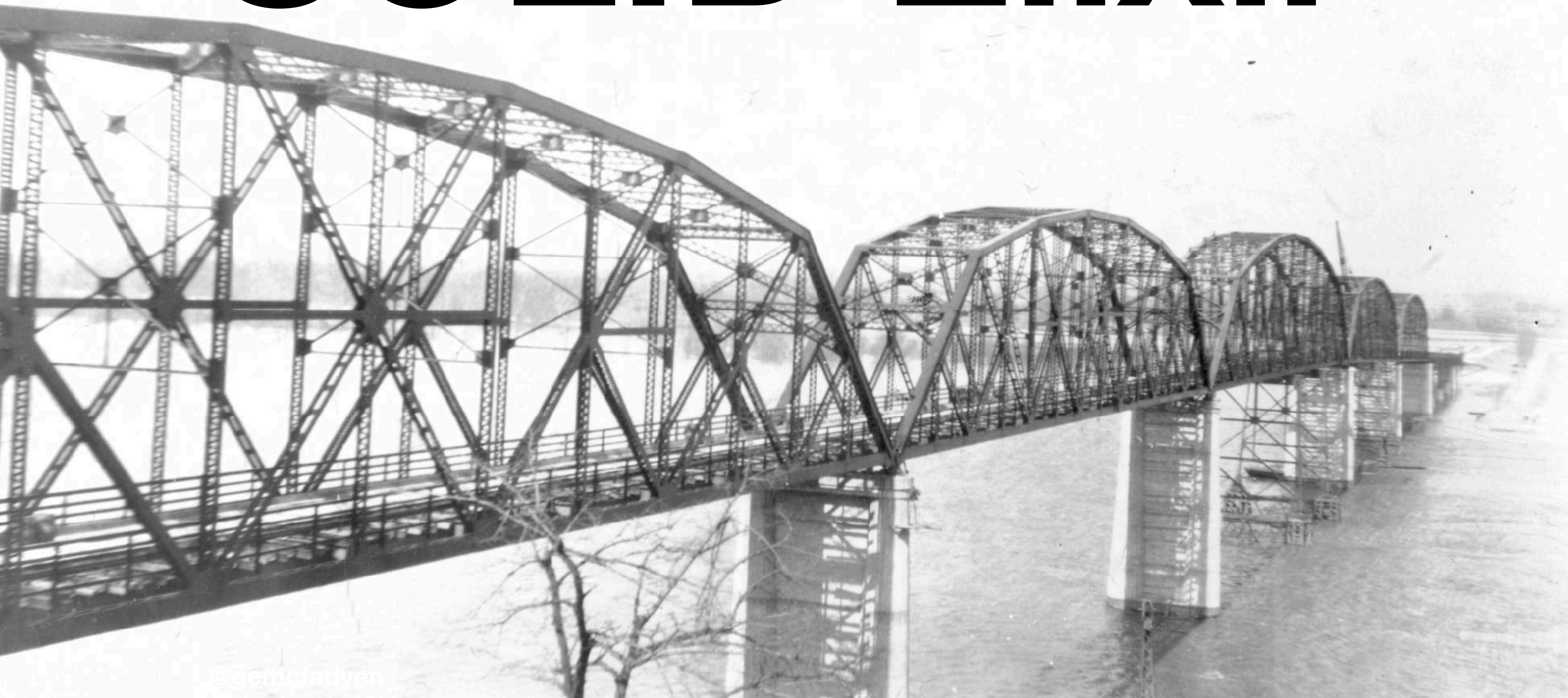
SOLID 

Abstractions

Immutability
Recursion
Pattern Matching
Currying

...

SOLID Elixir



S - - - Elixir



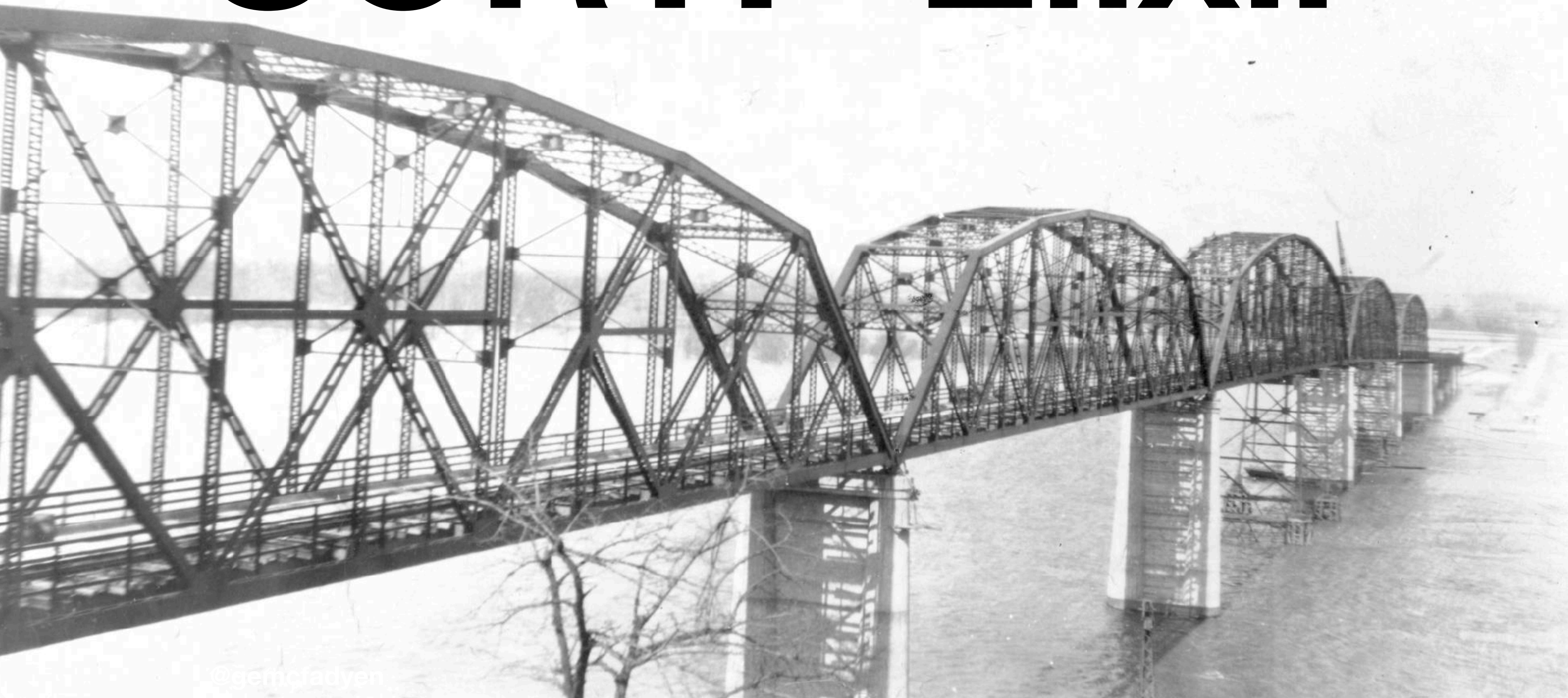
SO - - - Elixir



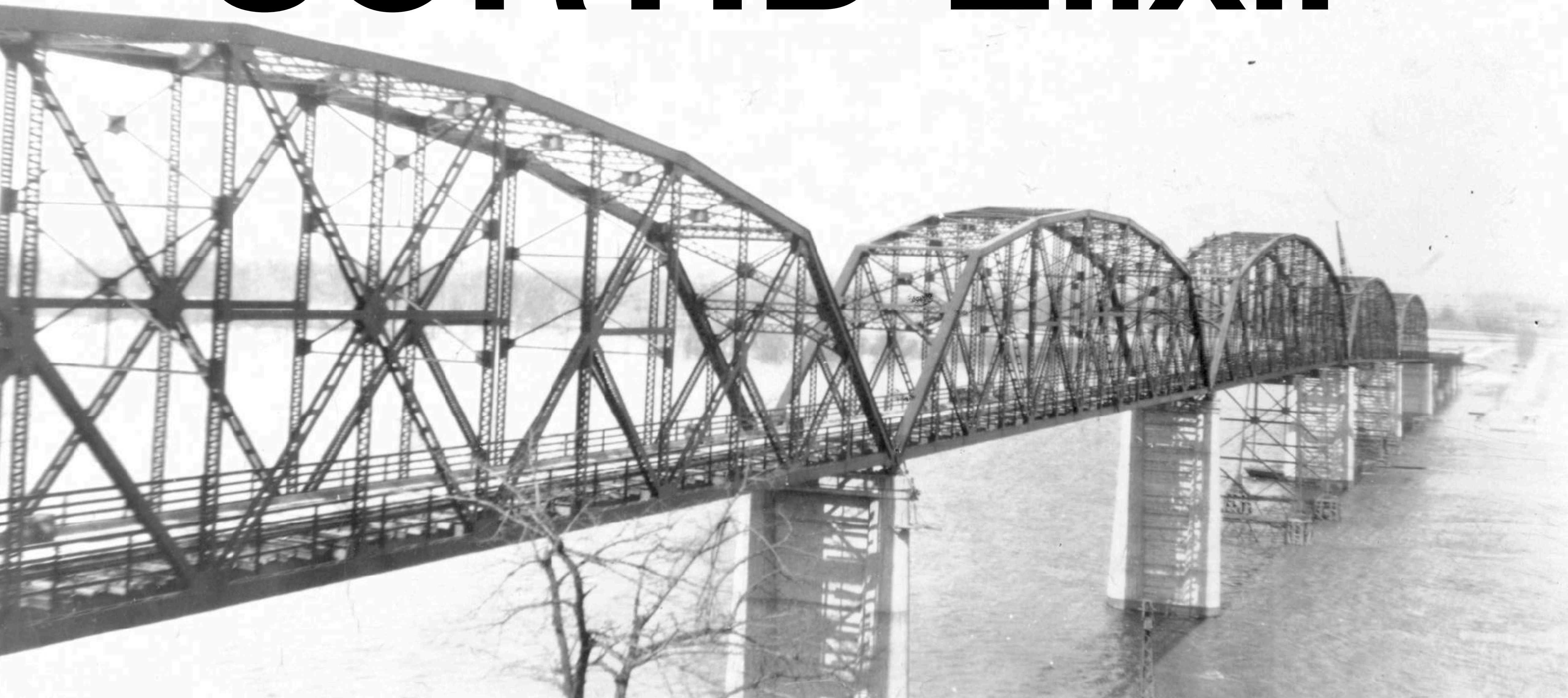
SORT- - Elixir



SORTI- Elixir



SORTID Elixir



Thank-you!



Please

**Remember to
rate this session**

Thank you!



Did you **remember**
to rate the previous
session ?

