WHO INSPIRES YOU?
#botjoy

WHAT'S STOPPING YOU?
yHirsch 10/16
botjoy

Event Sourcing with Commanded

Andriy Drozdyuk

Outline

Length: 30 minutes

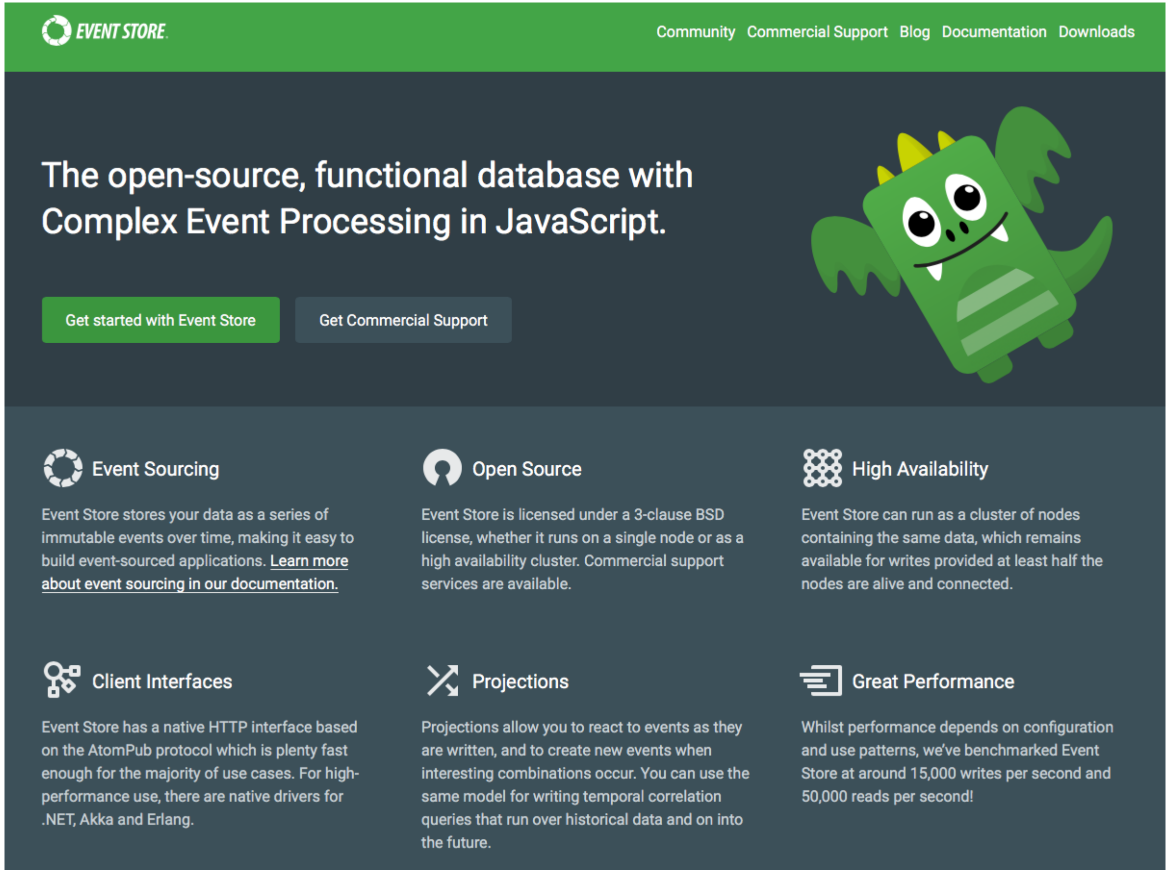
1. Preliminaries .
2. Commanded …
3. Design …
4. Code ………..
5. Demo …….
6. Issues .
7. Questions .
8. Homework .

Preliminaries

Event Store

Stores a series of events instead of final state.

Greg Young’s Event Store



The open-source, functional database with Complex Event Processing in JavaScript.

Event Sourcing

Event Store stores your data as a series of immutable events over time, making it easy to build event-sourced applications. Learn more about event sourcing in our documentation.

Open Source

Event Store is licensed under a 3-clause BSD license, whether it runs on a single node or as a high availability cluster. Commercial support services are available.

High Availability

Event Store can run as a cluster of nodes containing the same data, which remains available for writes provided at least half the nodes are alive and connected.

Client Interfaces

Event Store has a native HTTP interface based on the AtomPub protocol which is plenty fast enough for the majority of use cases. For high-performance use, there are native drivers for .NET, Akka and Erlang.

Projections

Projections allow you to react to events as they are written, and to create new events when interesting combinations occur. You can use the same model for writing temporal correlation queries that run over historical data and on into the future.

Great Performance

Whilst performance depends on configuration and use patterns, we’ve benchmarked Event Store at around 15,000 writes per second and 50,000 reads per second!

Aggregate

Model that represents your business logic.

For example, “Account” represents all the rules for depositing and withdrawing money.

Like a “class” but more general and domain specific.

Microservices meetup Oct 10, 2017

Event Sourcing

https://www.meetup.com/DDD-CQRS-ES/events/243443912/

Commanded

Hierarchy

Commanded

Hierarchy

$$\begin{tikzcd}\text{Commanded} \arrow[d, "\text{BEAM}"] \\ \text{Extreme}\end{tikzcd}$$

Hierarchy

$$\begin{tikzcd}
\text{Commanded} \arrow[d, "\text{BEAM}"] \\
\text{Extreme} \arrow[d, "\text{TCP}"] \\
\text{Event Store}
\end{tikzcd}$$

Hierarchy

$$\begin{tikzcd}
\text{Commanded} \arrow[d, "\text{BEAM}"] \\
\text{Extreme} \arrow[d, "\text{TCP}"] \\
\text{Event Store}
\end{tikzcd}$$

Greg Young's [Event Store]

Hierarchy

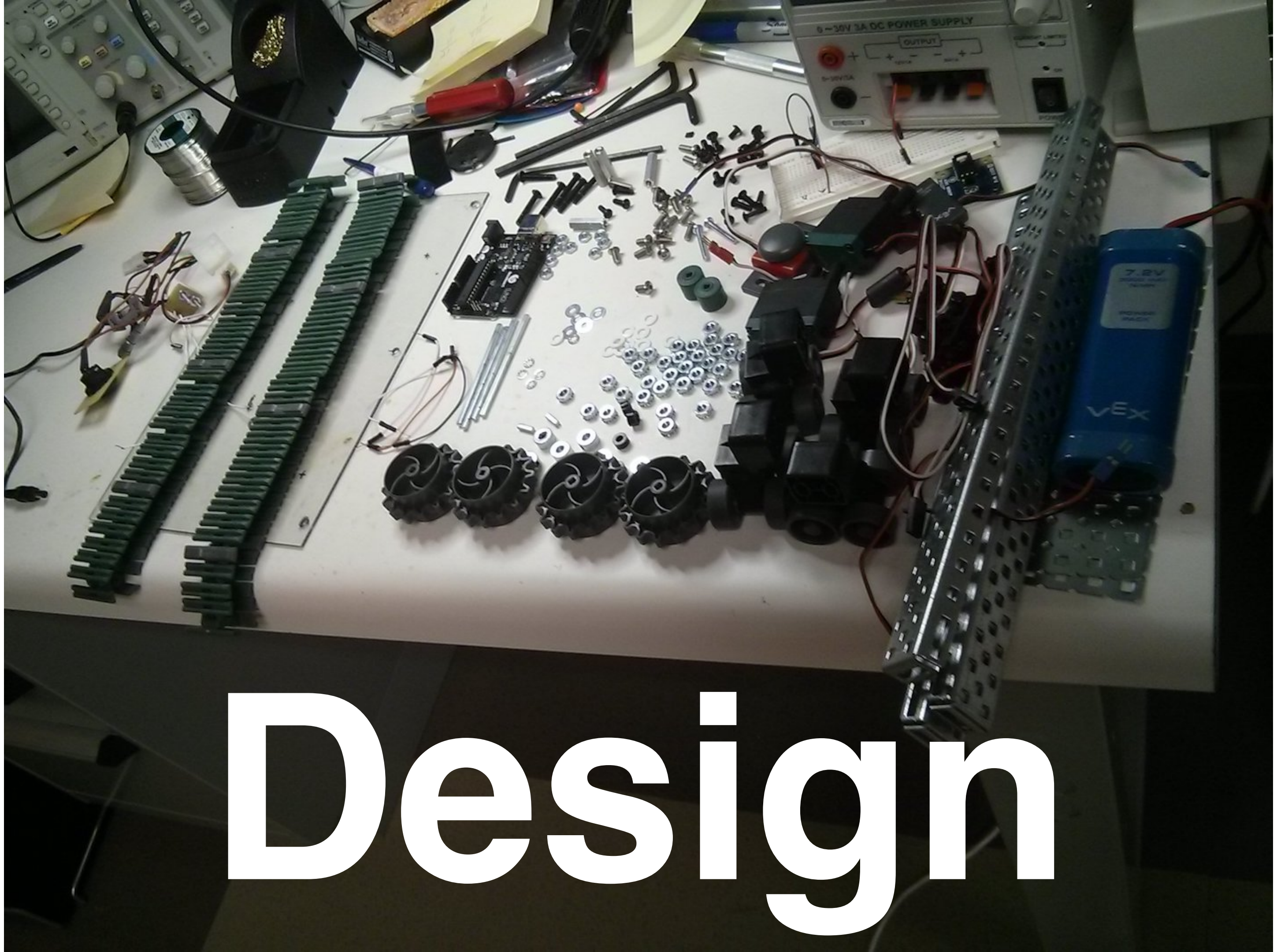
$$\begin{tikzcd}
\text{Commanded} \arrow[d, "\text{BEAM}"] & \\
\text{Extreme} \arrow[d, "\text{TCP}"] & \text{Abstracts to Elixir processes} \\
\text{Event Store} & \text{Greg Young's Event Store}
\end{tikzcd}$$

Hierarchy

$$\begin{tikzcd}
\text{Commanded} \arrow[d, "\text{BEAM}"] & \text{Abstracts to DDD} \\
\text{Extreme} \arrow[d, "\text{TCP}"] & \text{Abstracts to Elixir processes} \\
\text{Event Store} & \text{Greg Young's Event Store}
\end{tikzcd}$$

Commanded is an Elixir library
for applying domain driven design
to event sourced systems.

</Commanded>



Design

Bank

Customer can open an account, deposit and withdraw money

Don’t allow withdrawals if not enough money on account

Commands?

Commands?

Open an account

Commands?

Open an account

Deposit money

Commands?

Open an account

Deposit money

Withdraw money

Commands?

Open an account

Deposit money

Withdraw money

Errors?

Commands?

Open an account

Deposit money

Withdraw money

Errors?

Already exists

Commands?

Open an account

Deposit money

Withdraw money

Errors?

Already exists

Invalid amount

Commands?	Errors?
Open an account	Already exists
Deposit money	Invalid amount
Withdraw money	Insufficient funds or invalid amount

Events?

Command ⇢ Event

Open an account

Deposit money

Withdraw money

Command	⇢	Event
Open an account		Account opened
Deposit money		
Withdraw money		

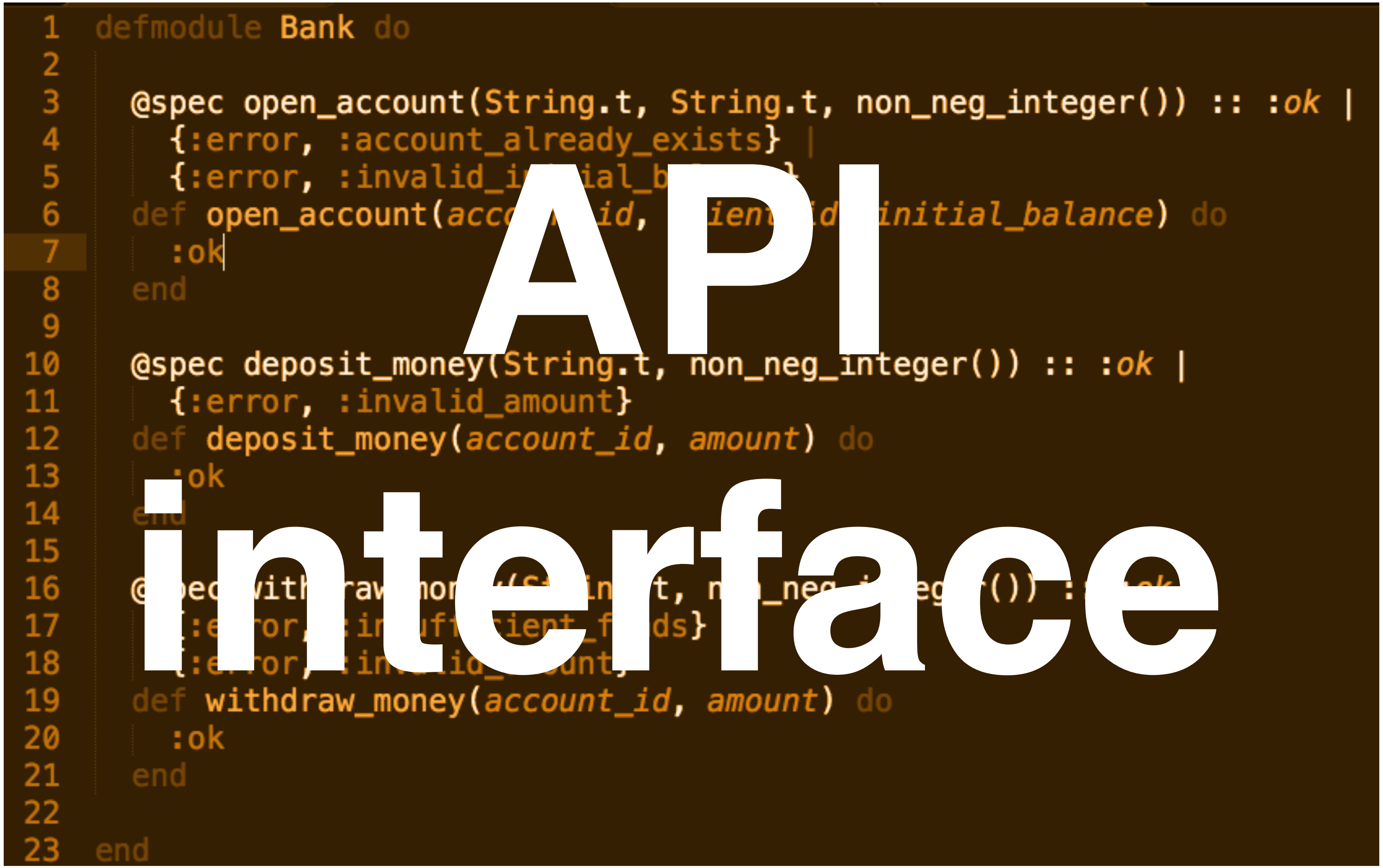
Command	⇢	Event
Open an account		Account opened
Deposit money		Money deposited
Withdraw money		

Command	⇢	Event
Open an account		Account opened
Deposit money		Money deposited
Withdraw money		Money withdrawn

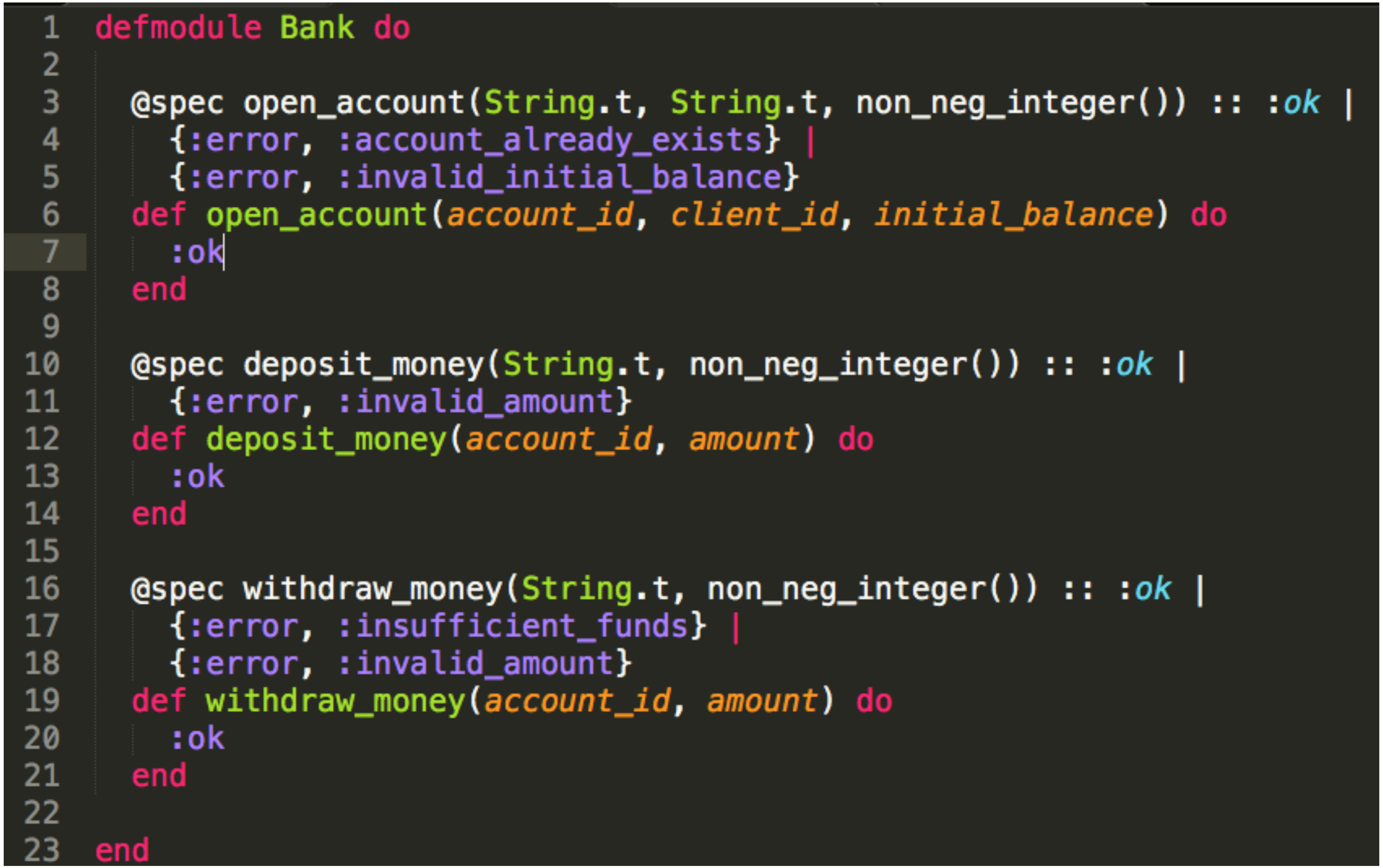
</Design>

Code

API interface

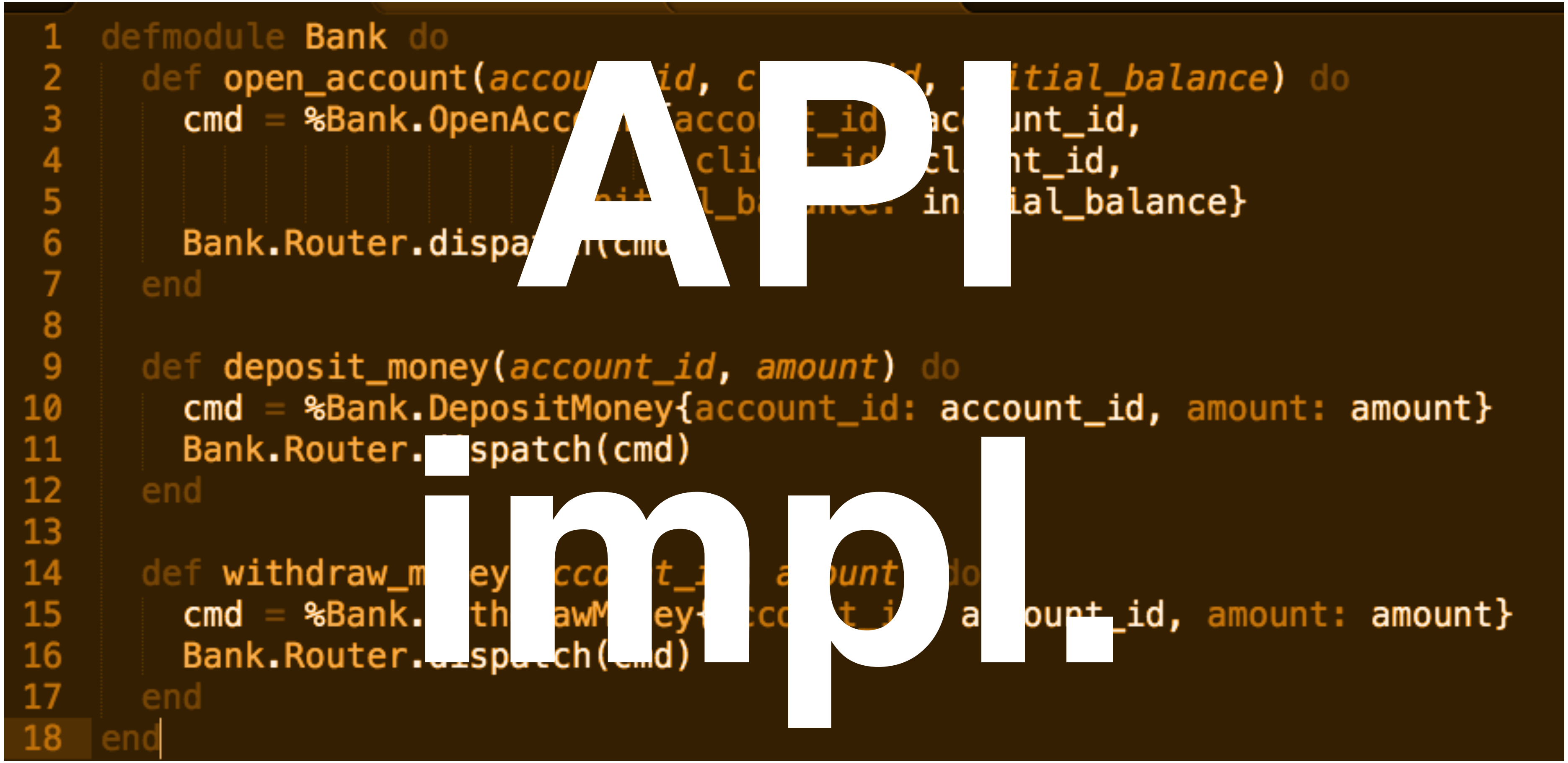


```
defmodule Bank do

  @spec open_account(String.t, String.t, non_neg_integer()) :: :ok |
    {:error, :account_already_exists} |
    {:error, :invalid_initial_balance}
  def open_account(account_id, client_id, initial_balance) do
    :ok
  end

  @spec deposit_money(String.t, non_neg_integer()) :: :ok |
    {:error, :invalid_amount}
  def deposit_money(account_id, amount) do
    :ok
  end

  @spec withdraw_money(String.t, non_neg_integer()) :: :ok |
    {:error, :insufficient_funds} |
    {:error, :invalid_amount}
  def withdraw_money(account_id, amount) do
    :ok
  end

end
```

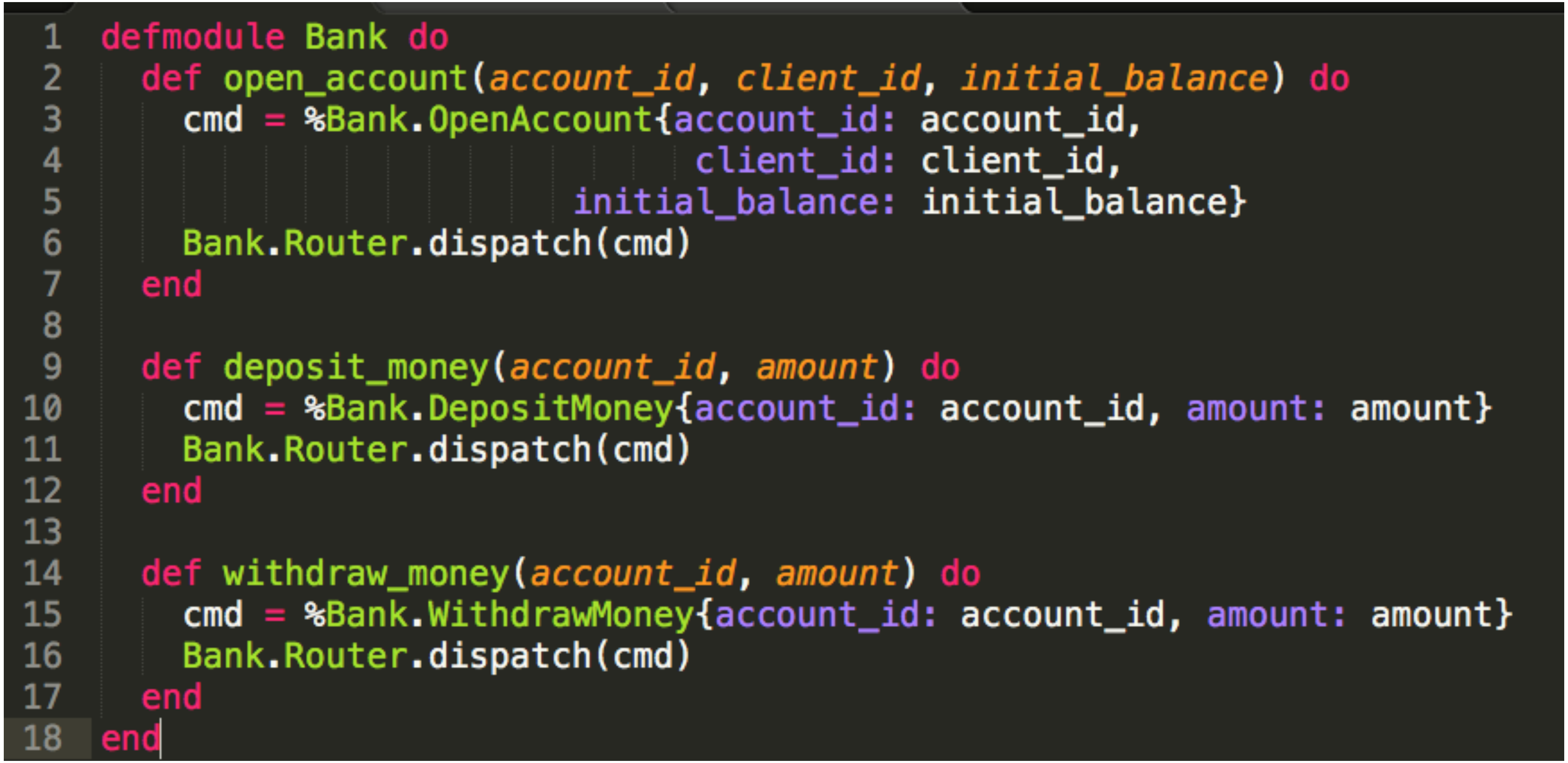


```elixir
defmodule Bank do

  @spec open_account(String.t, String.t, non_neg_integer()) :: :ok |
    {:error, :account_already_exists} |
    {:error, :invalid_initial_balance}
  def open_account(account_id, client_id, initial_balance) do
    :ok
  end

  @spec deposit_money(String.t, non_neg_integer()) :: :ok |
    {:error, :invalid_amount}
  def deposit_money(account_id, amount) do
    :ok
  end

  @spec withdraw_money(String.t, non_neg_integer()) :: :ok |
    {:error, :insufficient_funds} |
    {:error, :invalid_amount}
  def withdraw_money(account_id, amount) do
    :ok
  end

end
```

API impl.

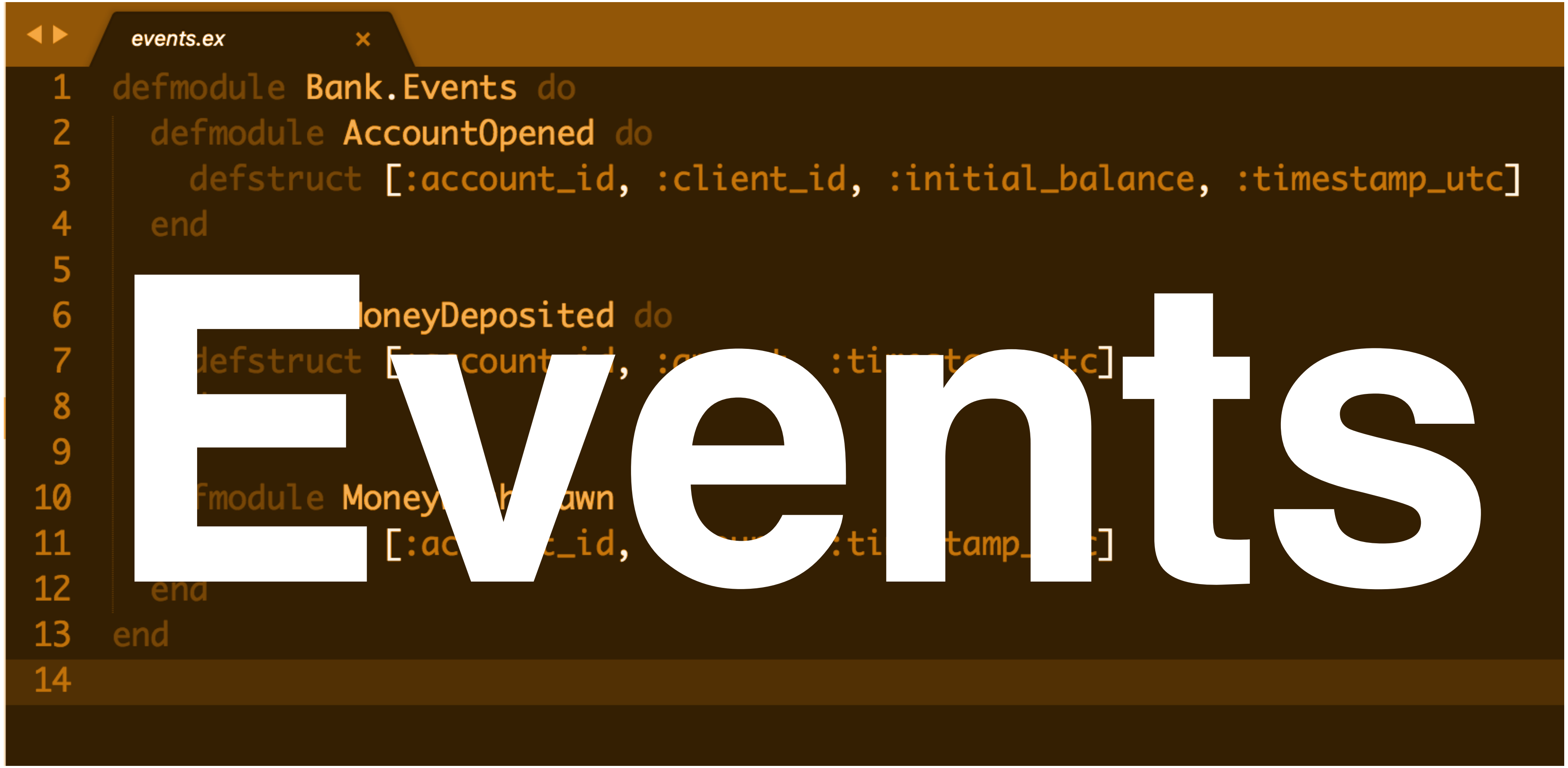


```
defmodule Bank do
  def open_account(accou[illegible]id, c[illegible]d, [illegible]itial_balance) do
    cmd = %Bank.OpenAcc[illegible]{acco[illegible]t_id[illegible] ac[illegible]unt_id,
                             cli[illegible]t_id[illegible] cl[illegible]nt_id,
                            [illegible]ial_b[illegible]nce: in[illegible]ial_balance}
    Bank.Router.dispa[illegible](cm[illegible]
  end

  def deposit_money(account_id, amount) do
    cmd = %Bank.DepositMoney{account_id: account_id, amount: amount}
    Bank.Router.[illegible]spatch(cmd)
  end

  def withdraw_m[illegible]ey[illegible]ccou[illegible]t_i[illegible] a[illegible]ount) do
    cmd = %Bank.[illegible]th[illegible]awM[illegible]ey{[illegible]cco[illegible]t_i[illegible] a[illegible]oun[illegible]_id, amount: amount}
    Bank.Router.[illegible]sp[illegible]ch([illegible]d)
  end
end
```

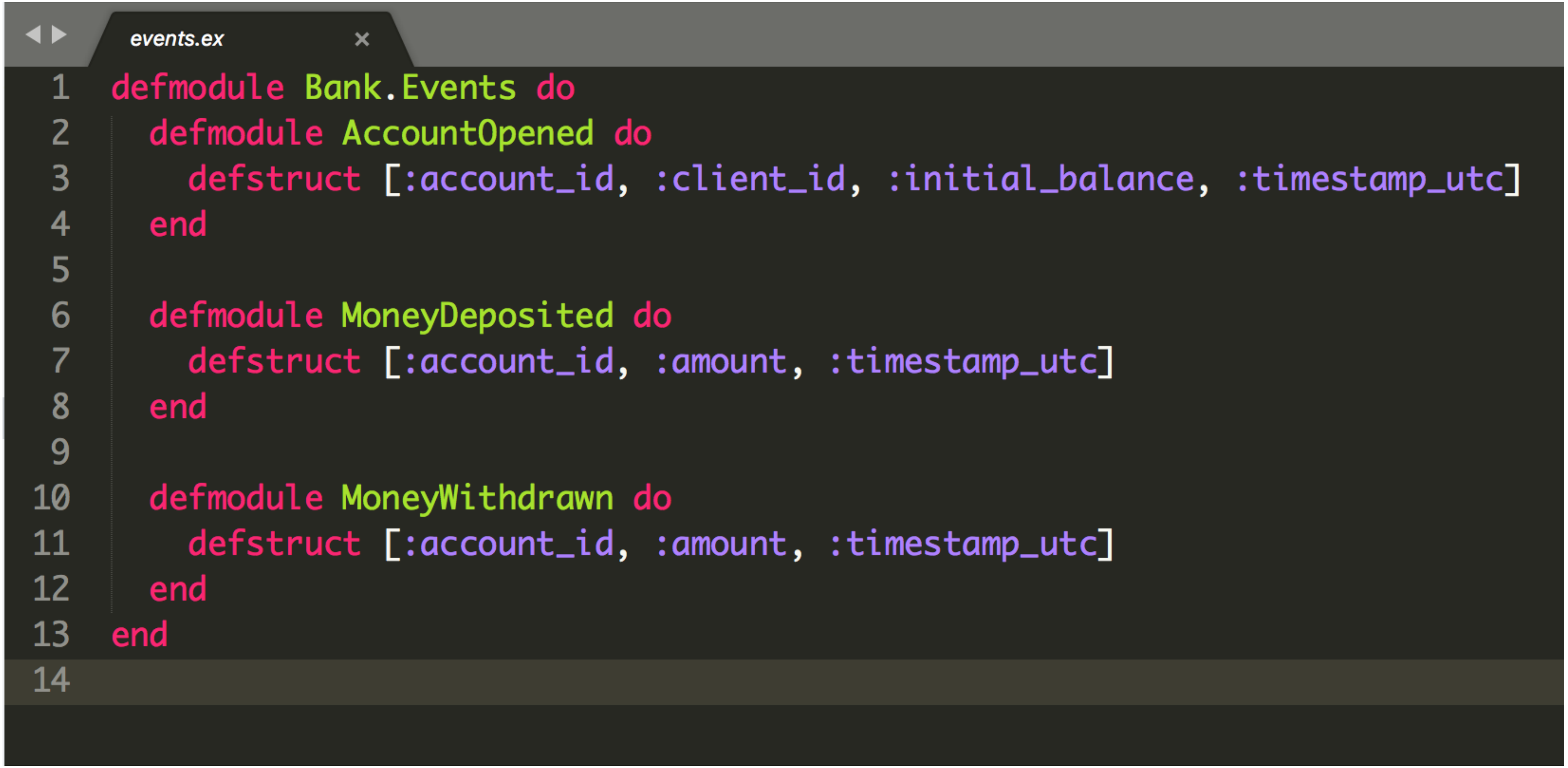


```elixir
defmodule Bank do
  def open_account(account_id, client_id, initial_balance) do
    cmd = %Bank.OpenAccount{account_id: account_id,
                             client_id: client_id,
                       initial_balance: initial_balance}
    Bank.Router.dispatch(cmd)
  end

  def deposit_money(account_id, amount) do
    cmd = %Bank.DepositMoney{account_id: account_id, amount: amount}
    Bank.Router.dispatch(cmd)
  end

  def withdraw_money(account_id, amount) do
    cmd = %Bank.WithdrawMoney{account_id: account_id, amount: amount}
    Bank.Router.dispatch(cmd)
  end
end
```

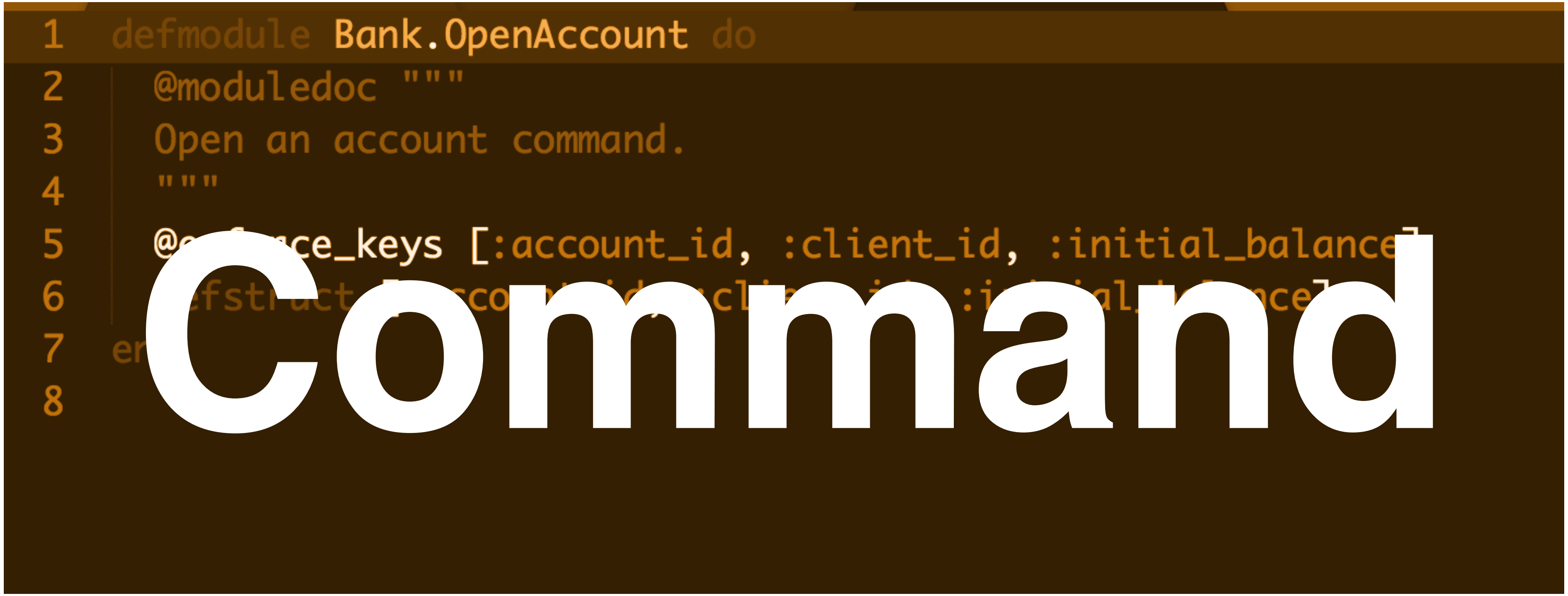
Events



```
defmodule Bank.Events do
  defmodule AccountOpened do
    defstruct [:account_id, :client_id, :initial_balance, :timestamp_utc]
  end

          oneyDeposited do
     efstruct [  count  , :a      :ti        tc]


   fmodule Money    h  awn
             [:ac    _id,         :ti    tamp_  c]
  end
end
```

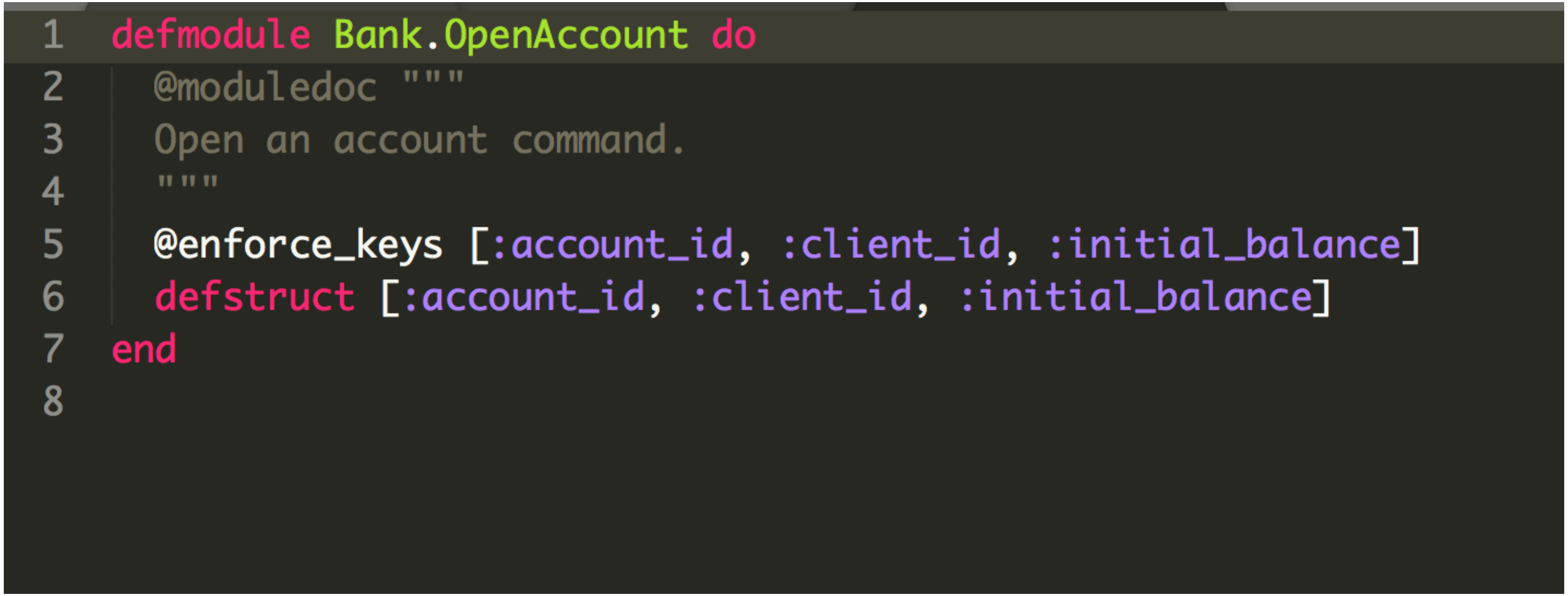

events.ex

```elixir
defmodule Bank.Events do
  defmodule AccountOpened do
    defstruct [:account_id, :client_id, :initial_balance, :timestamp_utc]
  end

  defmodule MoneyDeposited do
    defstruct [:account_id, :amount, :timestamp_utc]
  end

  defmodule MoneyWithdrawn do
    defstruct [:account_id, :amount, :timestamp_utc]
  end
end
```



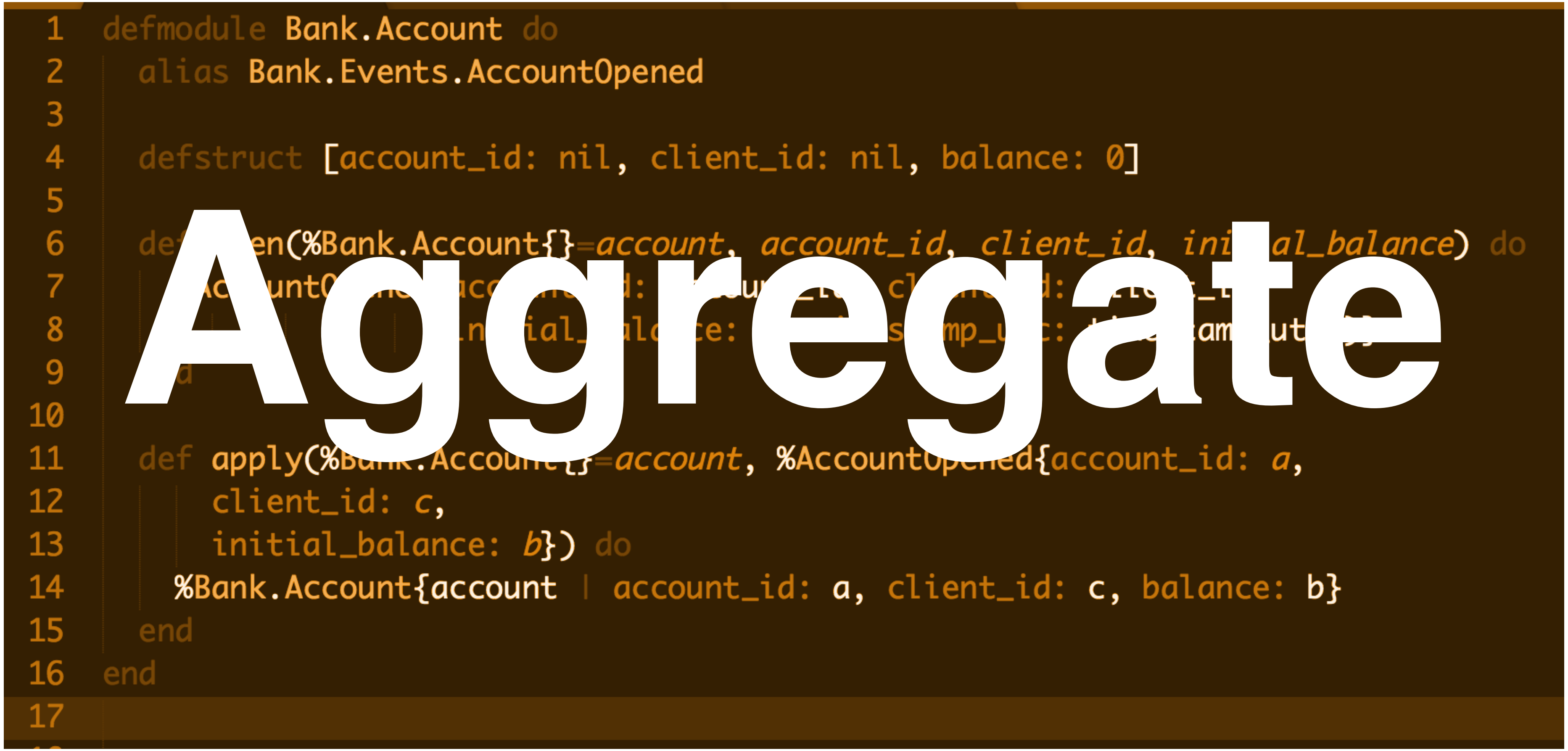
```elixir
defmodule Bank.OpenAccount do
  @moduledoc """
  Open an account command.
  """
  @enforce_keys [:account_id, :client_id, :initial_balance]
  defstruct [:account_id, :client_id, :initial_balance]
end
```

Command

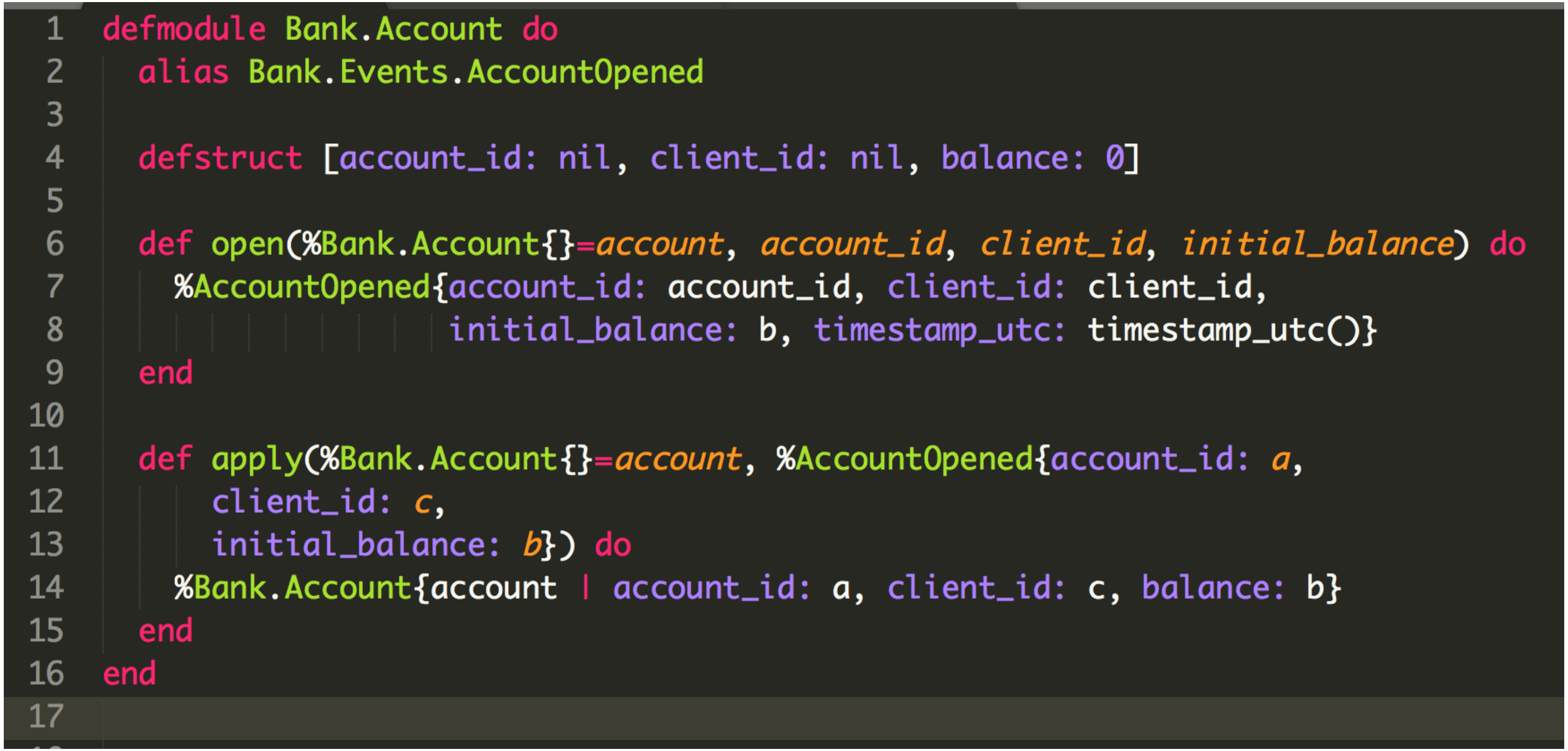


```elixir
defmodule Bank.OpenAccount do
  @moduledoc """
  Open an account command.
  """
  @enforce_keys [:account_id, :client_id, :initial_balance]
  defstruct [:account_id, :client_id, :initial_balance]
end
```

Aggregate



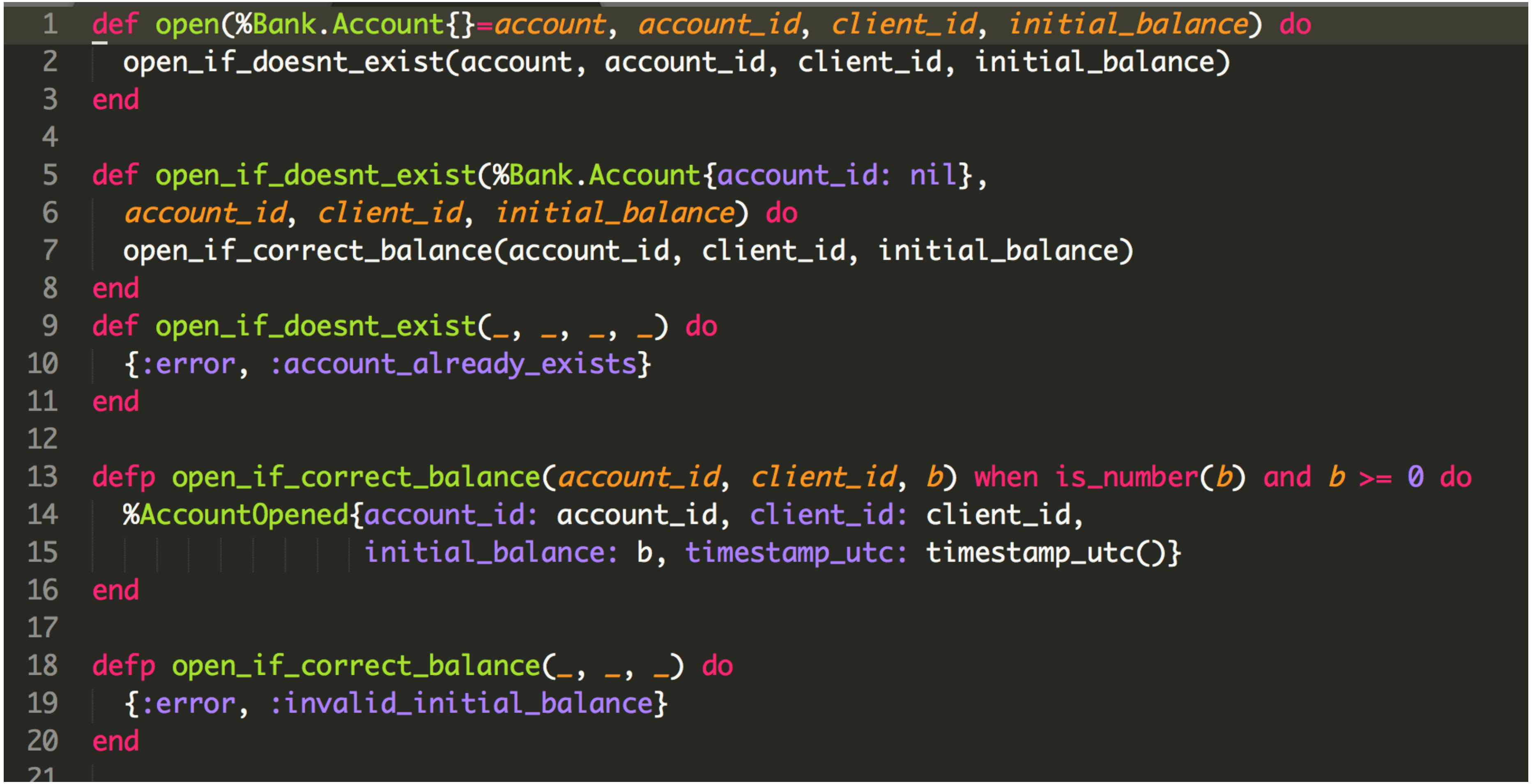
```
defmodule Bank.Account do
  alias Bank.Events.AccountOpened

  defstruct [account_id: nil, client_id: nil, balance: 0]

  de[illegible]en(%Bank.Account{}=account, account_id, client_id, ini[illegible]al_balance) do
    [illegible]
    [illegible]
  [illegible]d

  def apply(%Bank.Account{}=account, %AccountOpened{account_id: a,
      client_id: c,
      initial_balance: b}) do
    %Bank.Account{account | account_id: a, client_id: c, balance: b}
  end
end
```



```elixir
defmodule Bank.Account do
  alias Bank.Events.AccountOpened

  defstruct [account_id: nil, client_id: nil, balance: 0]

  def open(%Bank.Account{}=account, account_id, client_id, initial_balance) do
    %AccountOpened{account_id: account_id, client_id: client_id,
                   initial_balance: b, timestamp_utc: timestamp_utc()}
  end

  def apply(%Bank.Account{}=account, %AccountOpened{account_id: a,
      client_id: c,
      initial_balance: b}) do
    %Bank.Account{account | account_id: a, client_id: c, balance: b}
  end
end

```

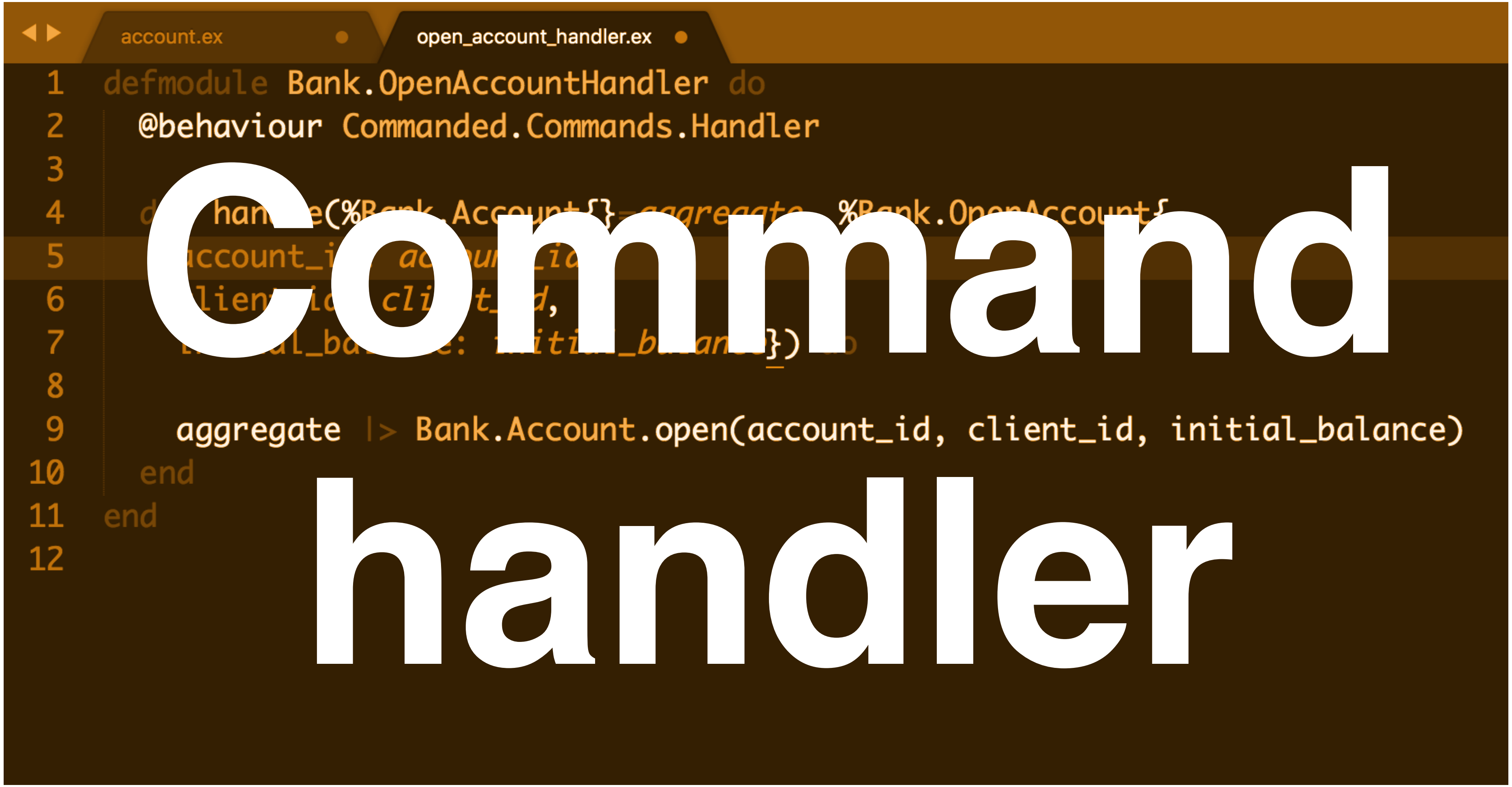
Error checking

```
def open(%Bank.Account{}=account, account_id, client_id, initial_balance) do
  open_if_doesnt_exist(account, account_id, client_id, initial_balance)
end

def open_if_doesnt_e[illegible]t(%Bank.Ac[illegible]{a[illegible]_id[illegible]
  account_id, client_[illegible]ial_[illegible]ance[illegible]
  open_if_correct_ba[illegible]ce(account[illegible]d, cl[illegible]t_id[illegible]niti[illegible]ba[illegible]ce)
end
def open_if_doesnt_e[illegible](_, _, _, _) do
  {:error, :account_already_exists}
end

defp open_if_[illegible]rrect_balance(account_id, clie[illegible]id, b) w[illegible] is_number(b) and b >= 0 do
  %AccountOpe[illegible]d{account_id: account_id, clie[illegible]id: client_id,
    [illegible]ial_[illegible]: b[illegible]amp[illegible]tc[illegible]ime[illegible]mp_[illegible]
e[illegible]

de[illegible]if_[illegible]rrec[illegible]bal[illegible]_, [illegible]
  {:error, :invalid_initial_balance}
end
```

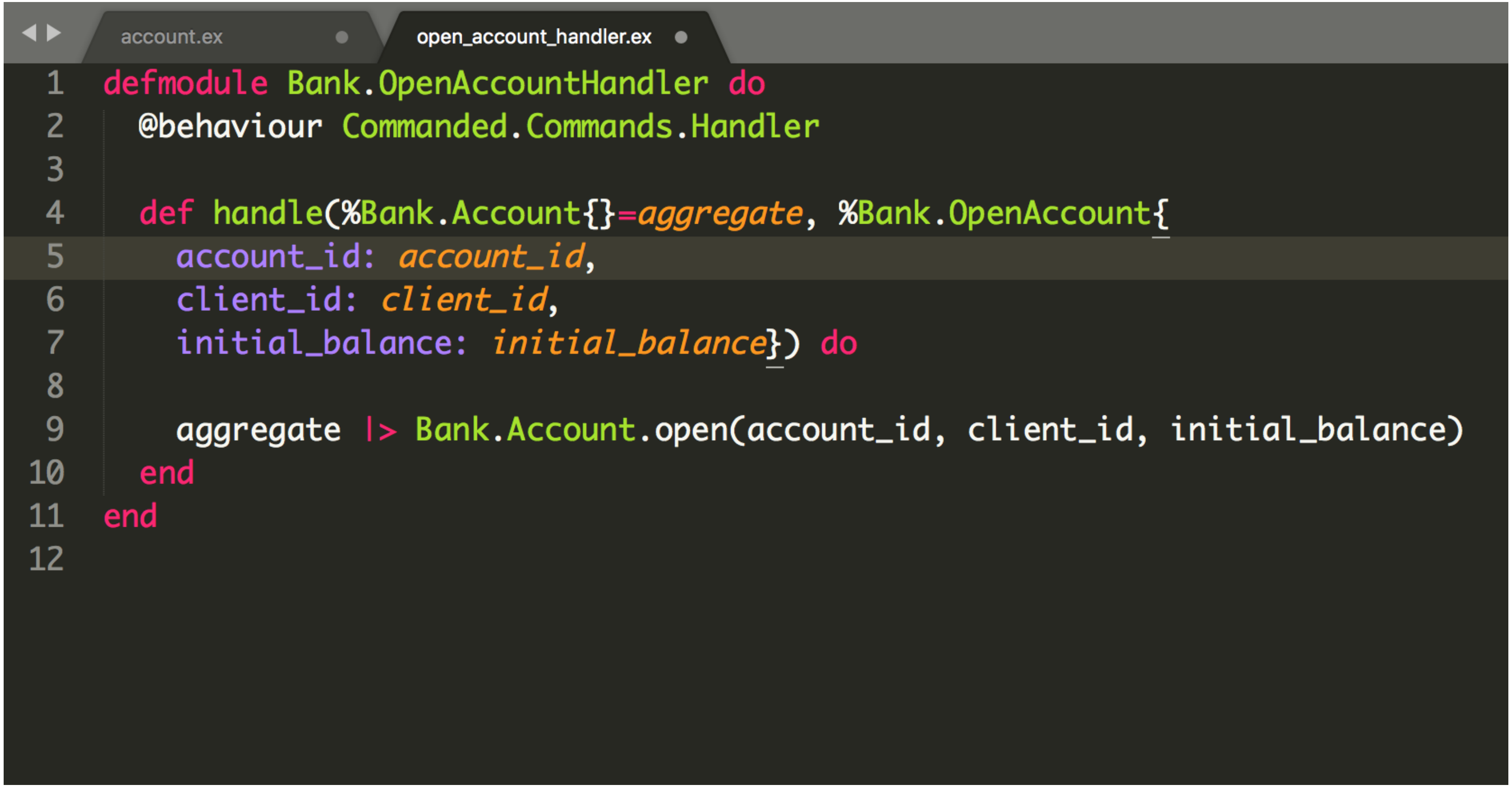


```elixir
def open(%Bank.Account{}=account, account_id, client_id, initial_balance) do
  open_if_doesnt_exist(account, account_id, client_id, initial_balance)
end

def open_if_doesnt_exist(%Bank.Account{account_id: nil},
  account_id, client_id, initial_balance) do
  open_if_correct_balance(account_id, client_id, initial_balance)
end
def open_if_doesnt_exist(_, _, _, _) do
  {:error, :account_already_exists}
end

defp open_if_correct_balance(account_id, client_id, b) when is_number(b) and b >= 0 do
  %AccountOpened{account_id: account_id, client_id: client_id,
                 initial_balance: b, timestamp_utc: timestamp_utc()}
end

defp open_if_correct_balance(_, _, _) do
  {:error, :invalid_initial_balance}
end
```

Command handler

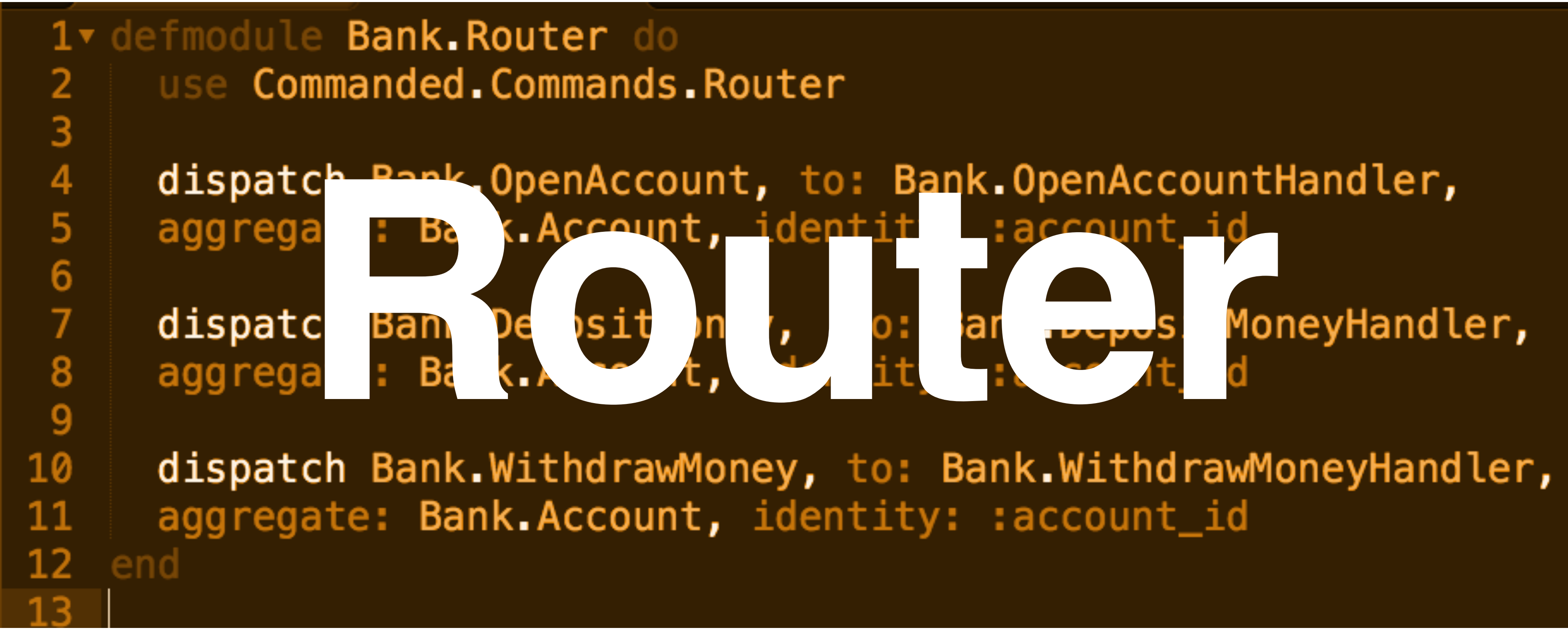

account.ex
open_account_handler.ex

```elixir
defmodule Bank.OpenAccountHandler do
  @behaviour Commanded.Commands.Handler

  def handle(%Bank.Account{} = aggregate, %Bank.OpenAccount{
    account_id: account_id,
    client_id: client_id,
    initial_balance: initial_balance}) do

    aggregate |> Bank.Account.open(account_id, client_id, initial_balance)
  end
end
```

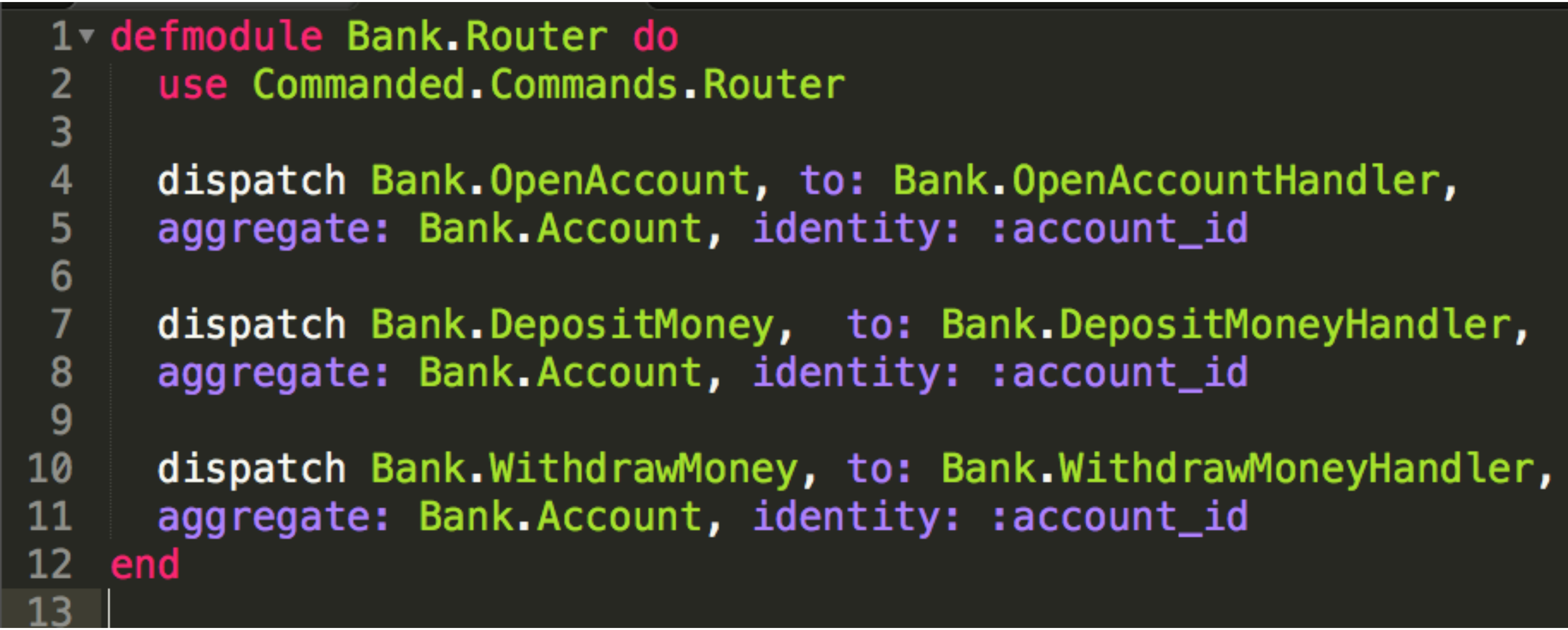


```elixir
defmodule Bank.OpenAccountHandler do
  @behaviour Commanded.Commands.Handler

  def handle(%Bank.Account{}=aggregate, %Bank.OpenAccount{
    account_id: account_id,
    client_id: client_id,
    initial_balance: initial_balance}) do

    aggregate |> Bank.Account.open(account_id, client_id, initial_balance)
  end
end

```

Router

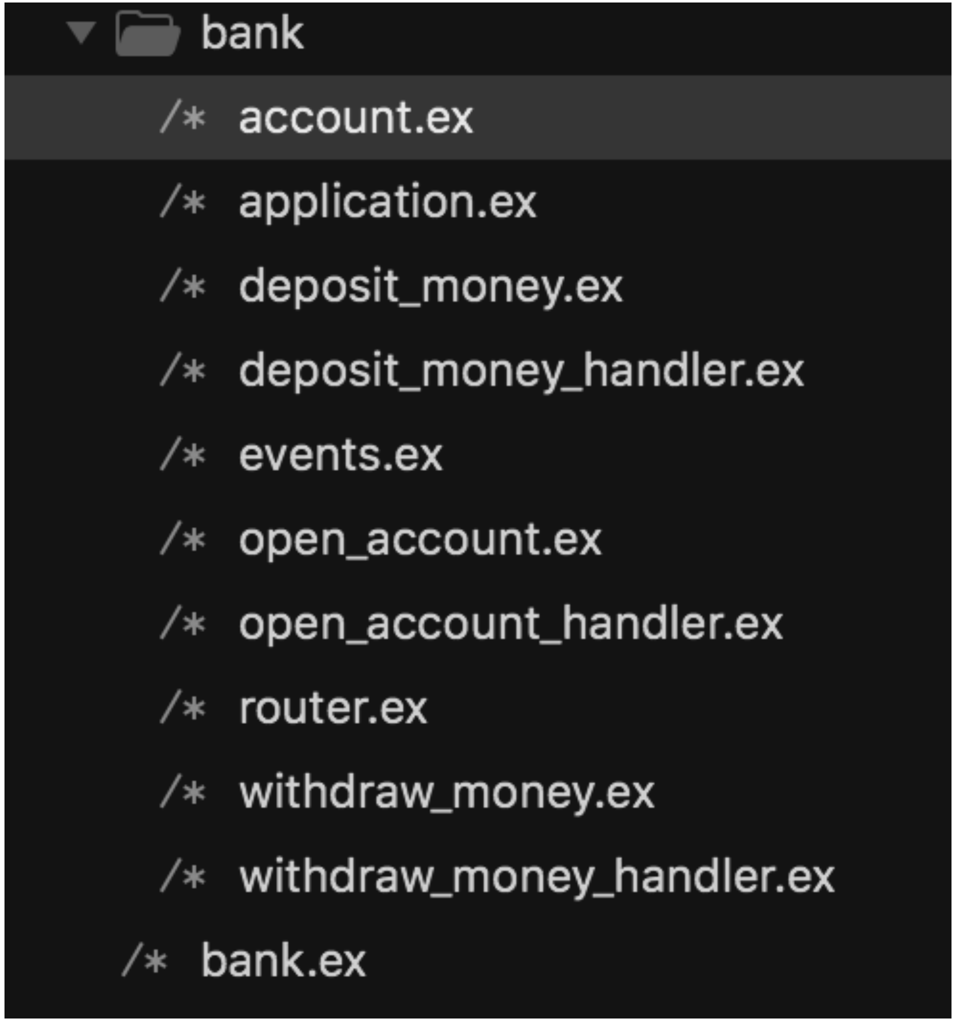


```elixir
defmodule Bank.Router do
  use Commanded.Commands.Router

  dispatch Bank.OpenAccount, to: Bank.OpenAccountHandler,
  aggregate: Bank.Account, identity: :account_id

  dispatch Bank.DepositMoney, to: Bank.DepositMoneyHandler,
  aggregate: Bank.Account, identity: :account_id

  dispatch Bank.WithdrawMoney, to: Bank.WithdrawMoneyHandler,
  aggregate: Bank.Account, identity: :account_id
end
```



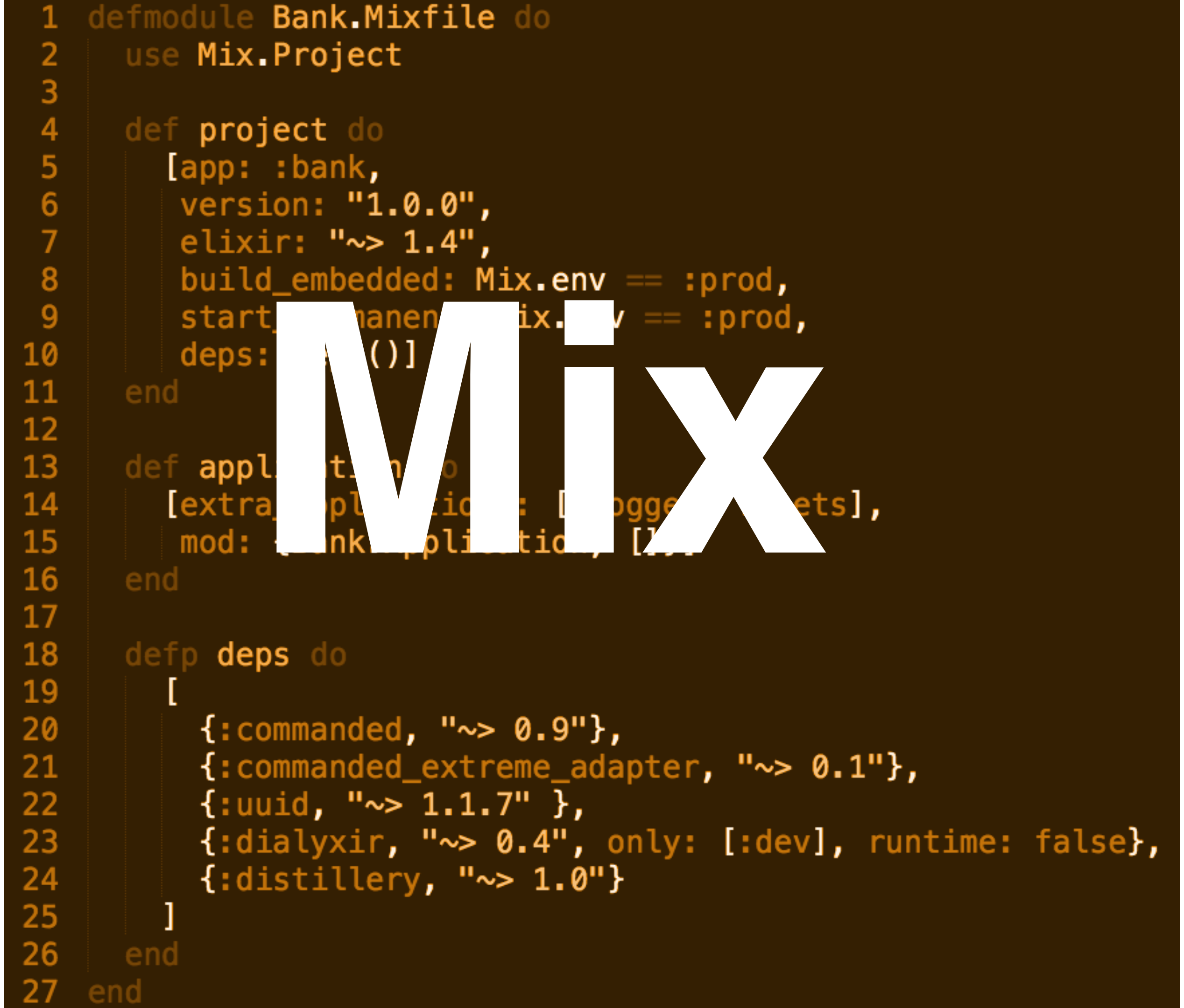
```elixir
defmodule Bank.Router do
  use Commanded.Commands.Router

  dispatch Bank.OpenAccount, to: Bank.OpenAccountHandler,
  aggregate: Bank.Account, identity: :account_id

  dispatch Bank.DepositMoney,  to: Bank.DepositMoneyHandler,
  aggregate: Bank.Account, identity: :account_id

  dispatch Bank.WithdrawMoney, to: Bank.WithdrawMoneyHandler,
  aggregate: Bank.Account, identity: :account_id
end
```

Rest

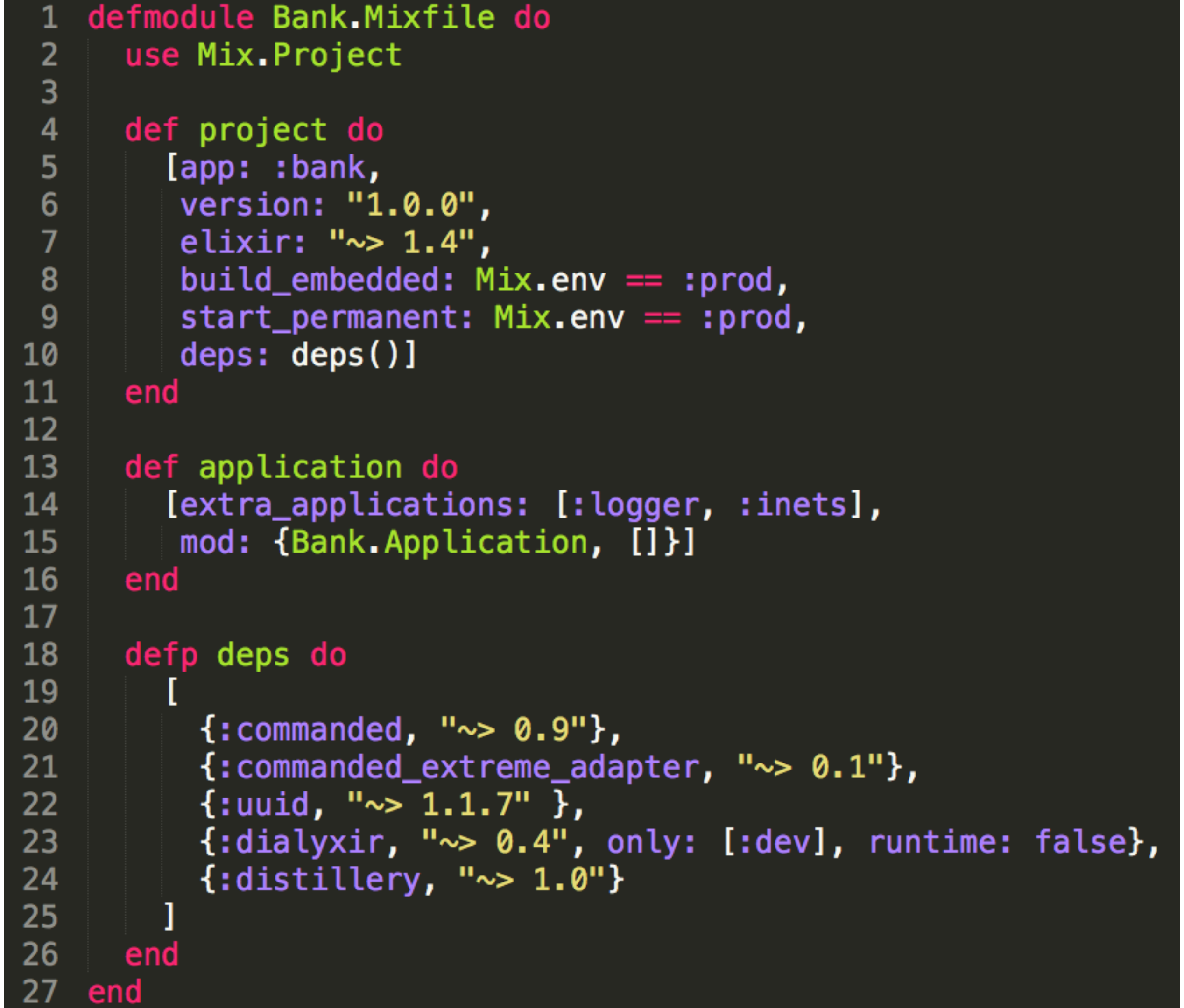


```
▼ bank
    /* account.ex
    /* application.ex
    /* deposit_money.ex
    /* deposit_money_handler.ex
    /* events.ex
    /* open_account.ex
    /* open_account_handler.ex
    /* router.ex
    /* withdraw_money.ex
    /* withdraw_money_handler.ex
  /* bank.ex
```

Mix



```
defmodule Bank.Mixfile do
  use Mix.Project

  def project do
    [app: :bank,
     version: "1.0.0",
     elixir: "~> 1.4",
     build_embedded: Mix.env == :prod,
     start_permanent: Mix.env == :prod,
     deps: deps()]
  end

  def application do
    [extra_applications: [:logger, :ets],
     mod: {Bank.Application, []}]
  end

  defp deps do
    [
      {:commanded, "~> 0.9"},
      {:commanded_extreme_adapter, "~> 0.1"},
      {:uuid, "~> 1.1.7" },
      {:dialyxir, "~> 0.4", only: [:dev], runtime: false},
      {:distillery, "~> 1.0"}
    ]
  end
end
```

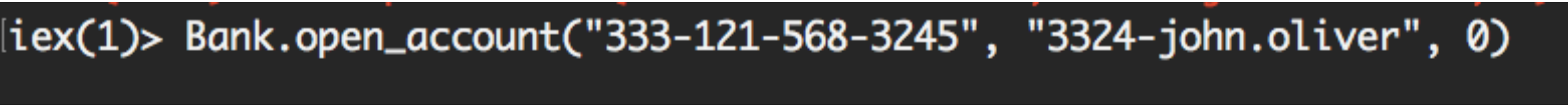


```elixir
defmodule Bank.Mixfile do
  use Mix.Project

  def project do
    [app: :bank,
     version: "1.0.0",
     elixir: "~> 1.4",
     build_embedded: Mix.env == :prod,
     start_permanent: Mix.env == :prod,
     deps: deps()]
  end

  def application do
    [extra_applications: [:logger, :inets],
     mod: {Bank.Application, []}]
  end

  defp deps do
    [
      {:commanded, "~> 0.9"},
      {:commanded_extreme_adapter, "~> 0.1"},
      {:uuid, "~> 1.1.7" },
      {:dialyxir, "~> 0.4", only: [:dev], runtime: false},
      {:distillery, "~> 1.0"}
    ]
  end
end
```

</Code>

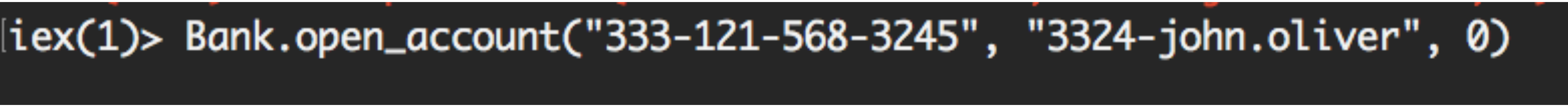
Demo

Open An Account

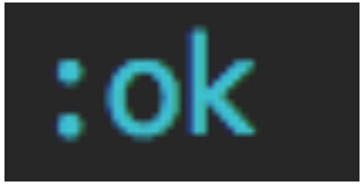


```
iex(1)> Bank.open_account("333-121-568-3245", "3324-john.oliver", 0)
```

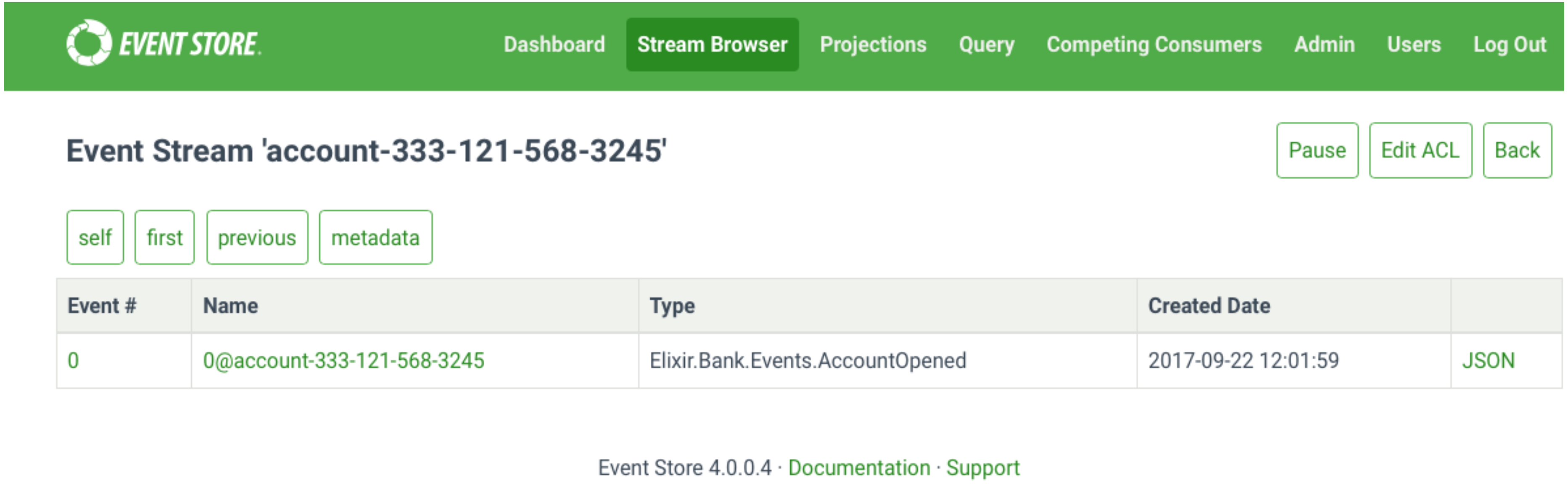
Open An Account



```
iex(1)> Bank.open_account("333-121-568-3245", "3324-john.oliver", 0)
```



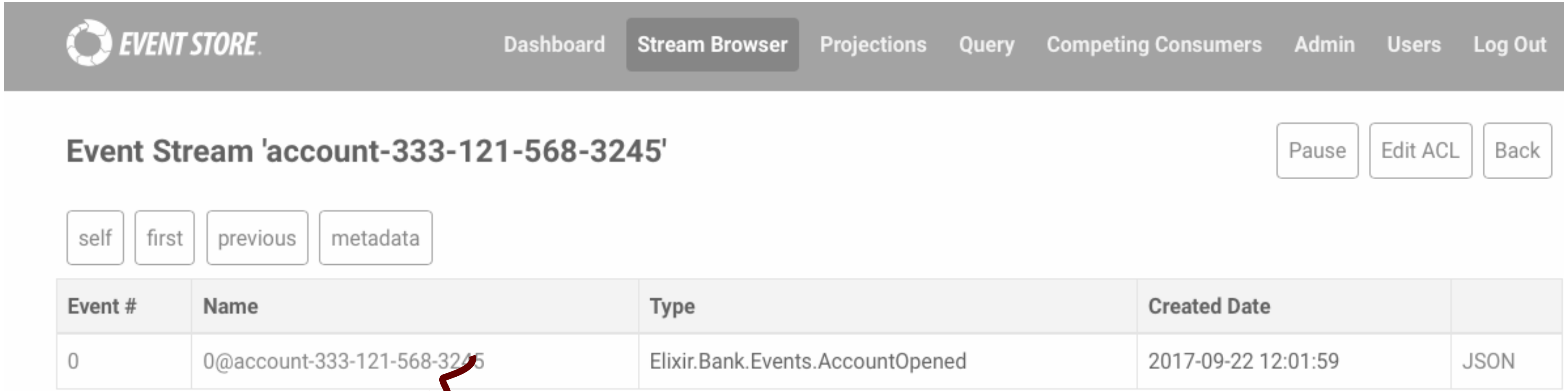
```
:ok
```



Event Stream 'account-333-121-568-3245'

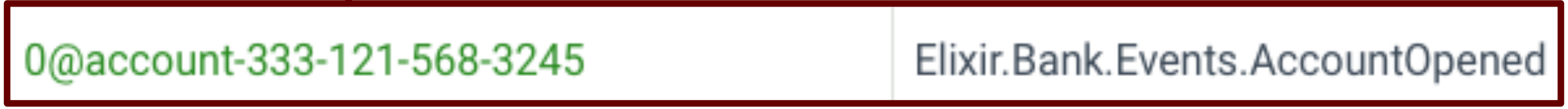
Event #	Name	Type	Created Date	
0	0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59	JSON

Event Store 4.0.0.4 · Documentation · Support

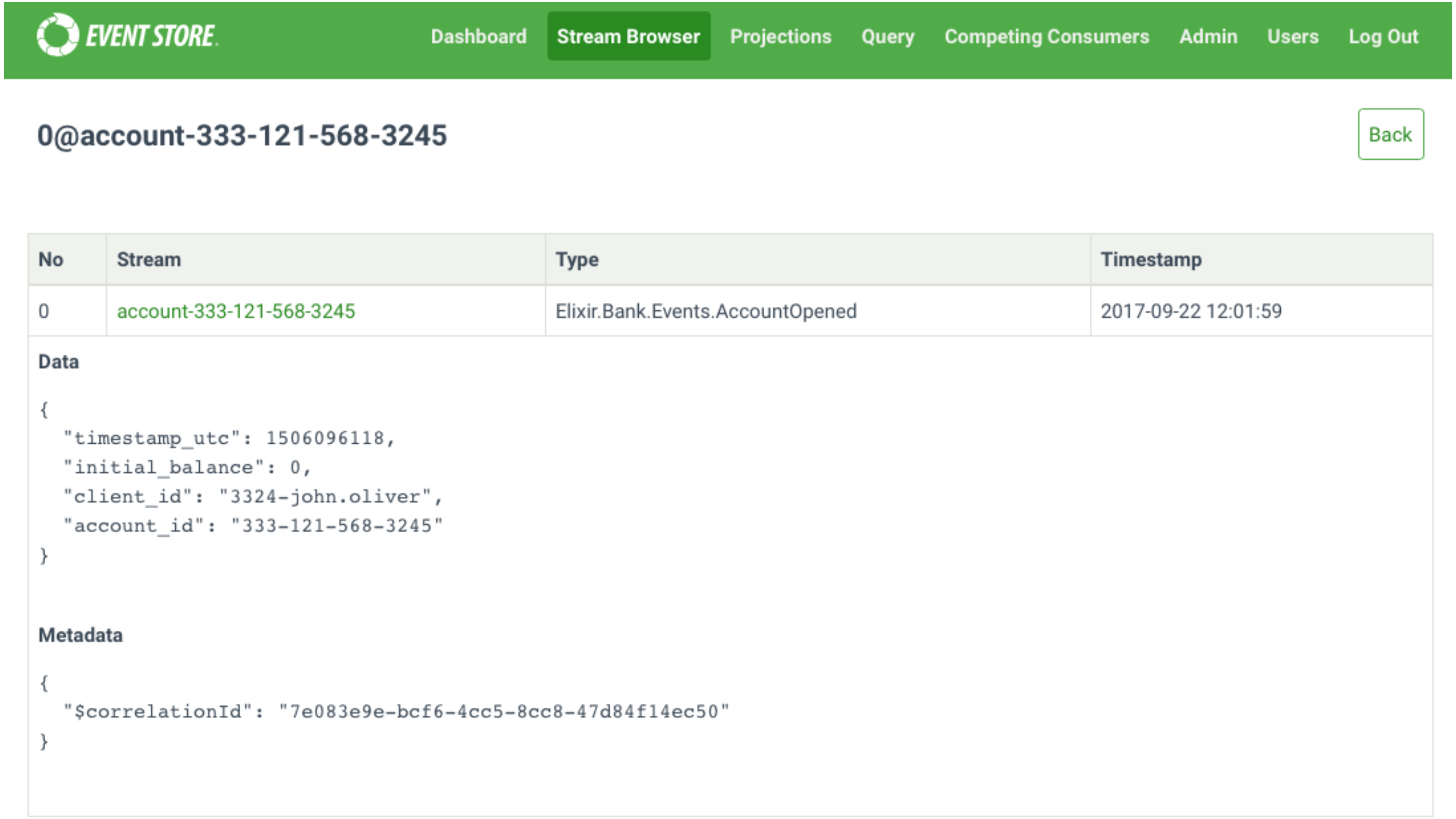


Event Stream 'account-333-121-568-3245'

Event #	Name	Type	Created Date	
0	0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59	JSON



0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened



0@account-333-121-568-3245

No	Stream	Type	Timestamp
0	account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59

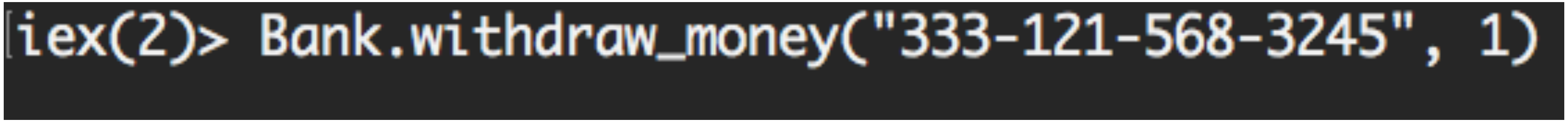
Data

```
{
  "timestamp_utc": 1506096118,
  "initial_balance": 0,
  "client_id": "3324-john.oliver",
  "account_id": "333-121-568-3245"
}
```

Metadata

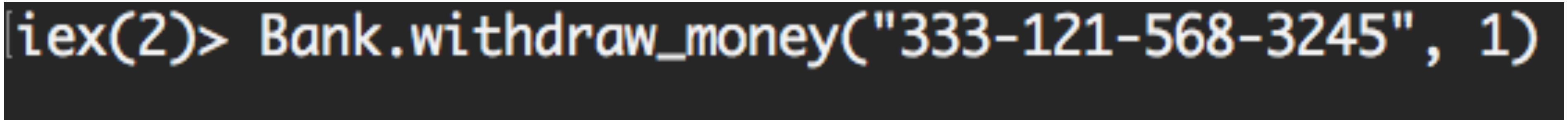
```
{
  "$correlationId": "7e083e9e-bcf6-4cc5-8cc8-47d84f14ec50"
}
```

Withdraw?

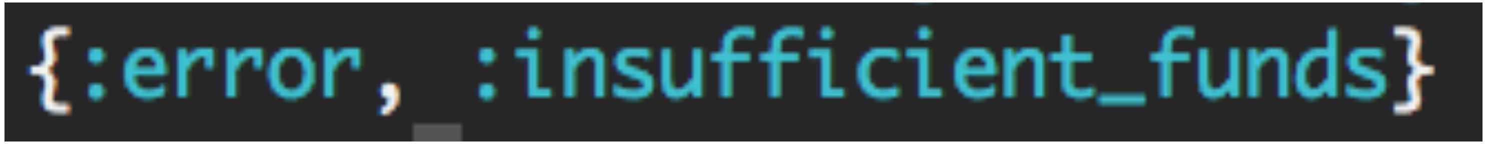


```
iex(2)> Bank.withdraw_money("333-121-568-3245", 1)
```

Withdraw?

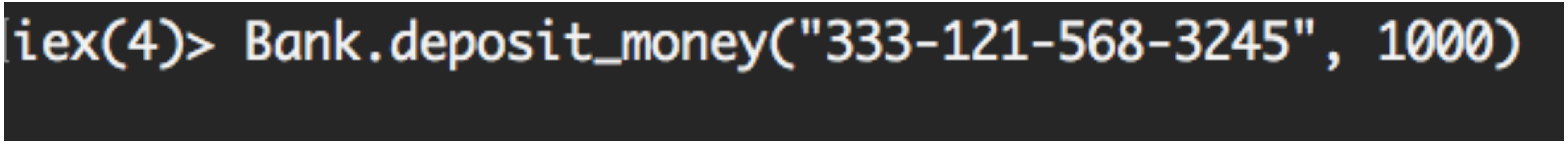


```
iex(2)> Bank.withdraw_money("333-121-568-3245", 1)
```



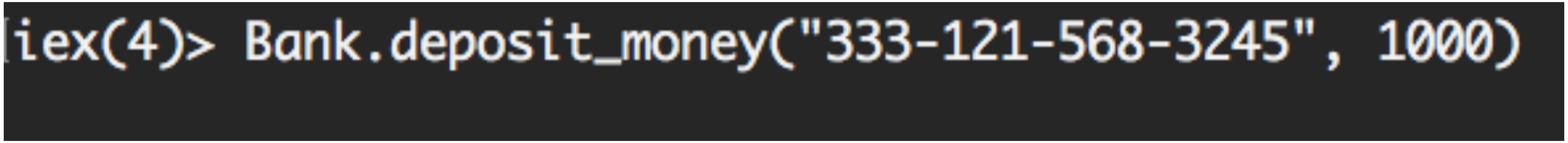
```
{:error, :insufficient_funds}
```

Deposit

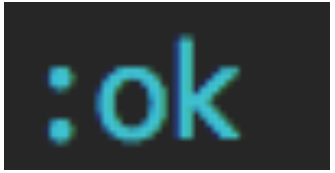


```
iex(4)> Bank.deposit_money("333-121-568-3245", 1000)
```

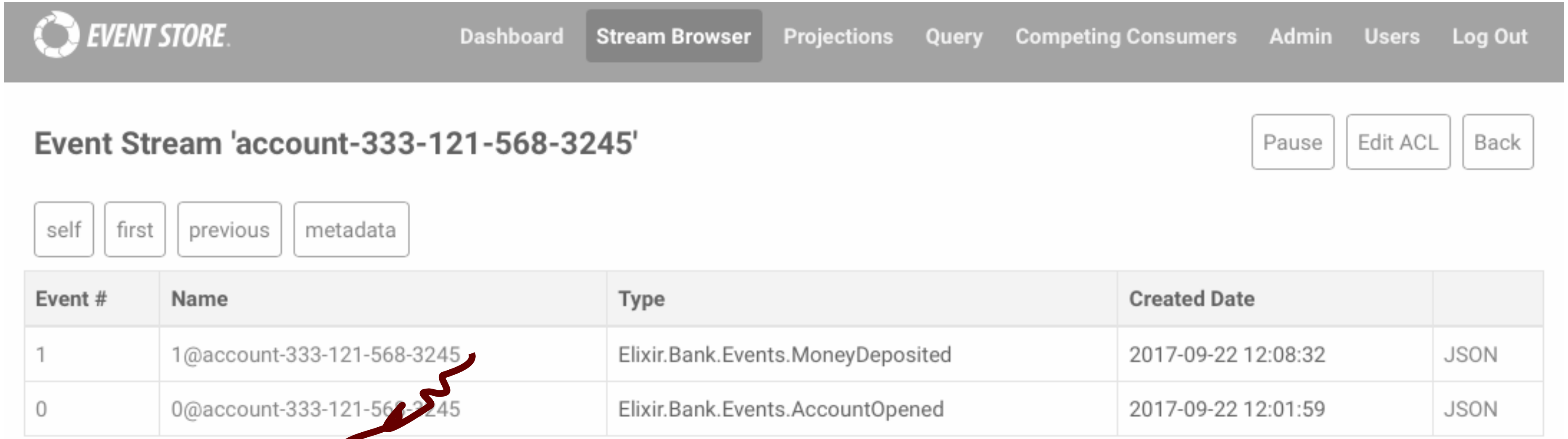
Deposit



```
iex(4)> Bank.deposit_money("333-121-568-3245", 1000)
```



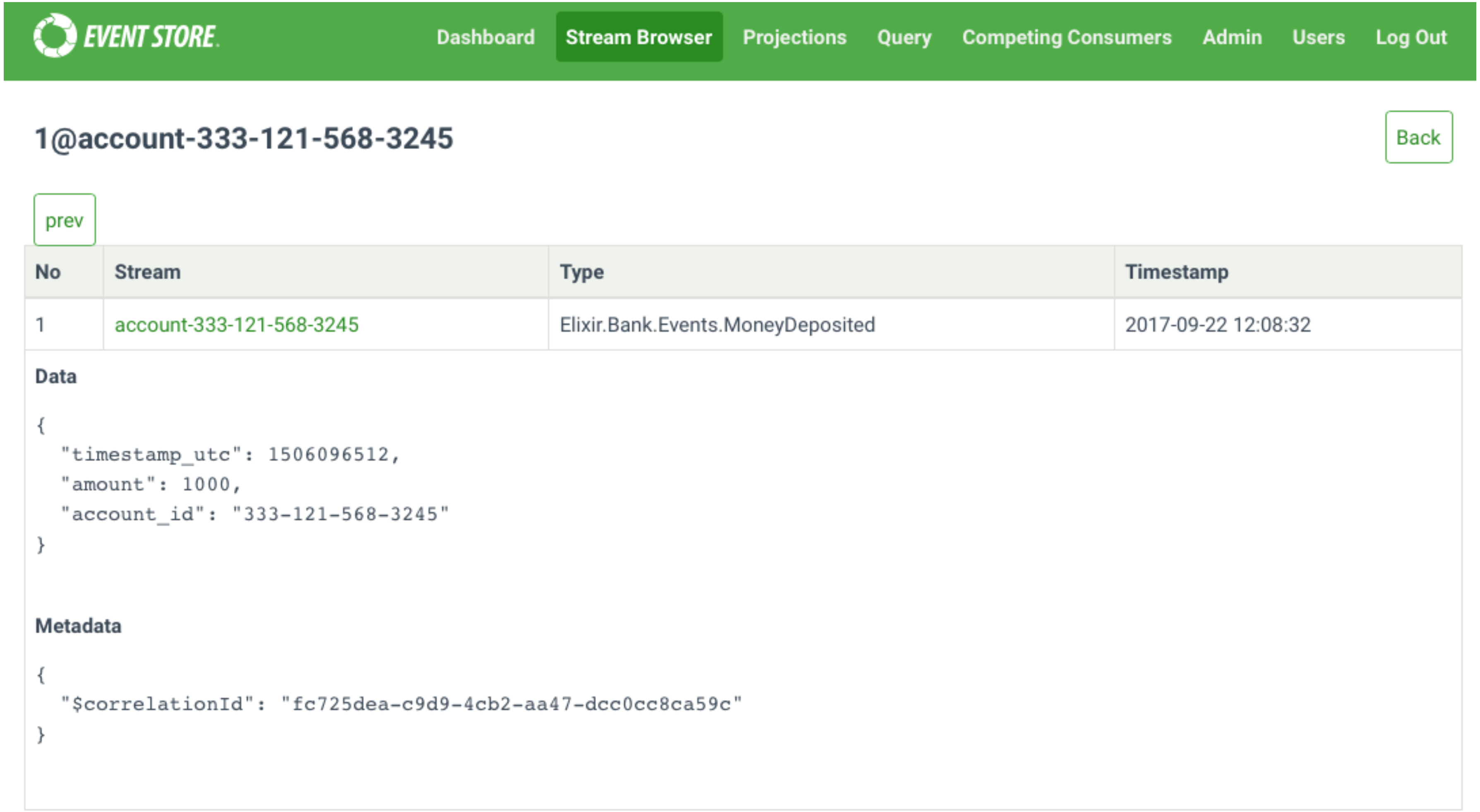
```
:ok
```



Event Stream 'account-333-121-568-3245'

Event #	Name	Type	Created Date	
1	1@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:08:32	JSON
0	0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59	JSON

1@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited



1@account-333-121-568-3245

No	Stream	Type	Timestamp
1	account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:08:32

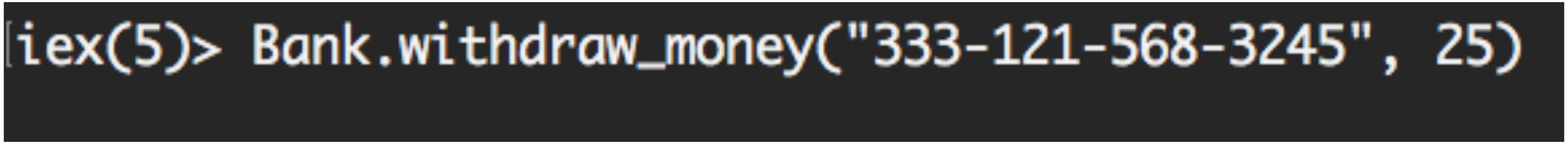
Data

```
{
  "timestamp_utc": 1506096512,
  "amount": 1000,
  "account_id": "333-121-568-3245"
}
```

Metadata

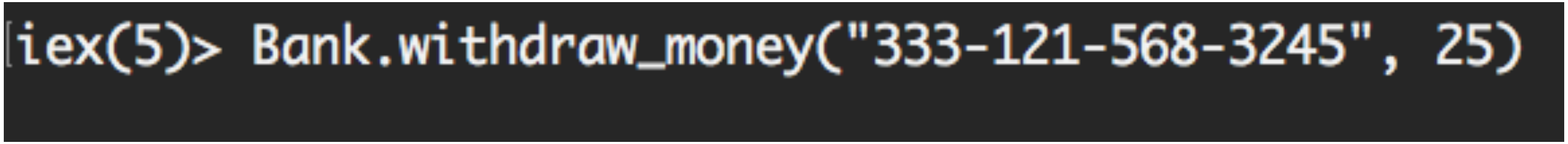
```
{
  "$correlationId": "fc725dea-c9d9-4cb2-aa47-dcc0cc8ca59c"
}
```

Try again?

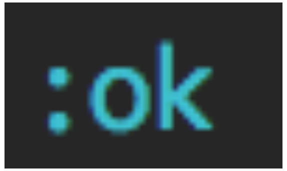


```
iex(5)> Bank.withdraw_money("333-121-568-3245", 25)
```

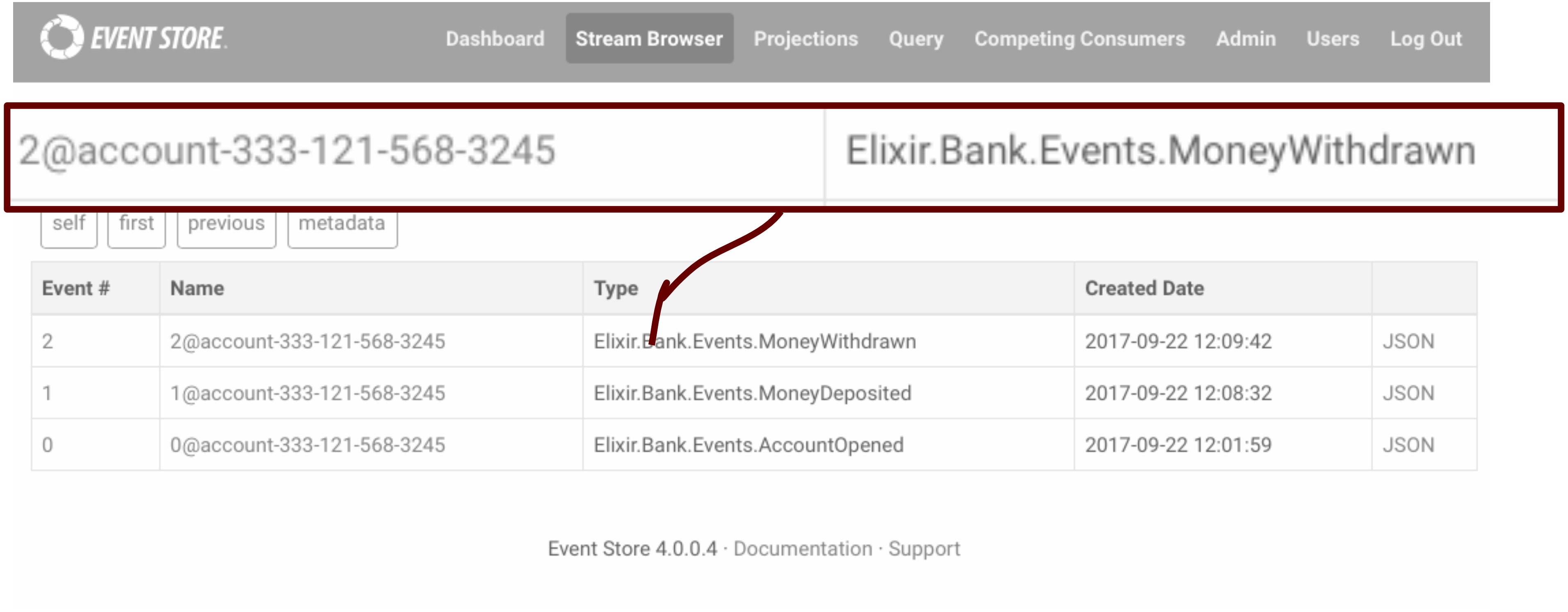
Try again?



```
iex(5)> Bank.withdraw_money("333-121-568-3245", 25)
```



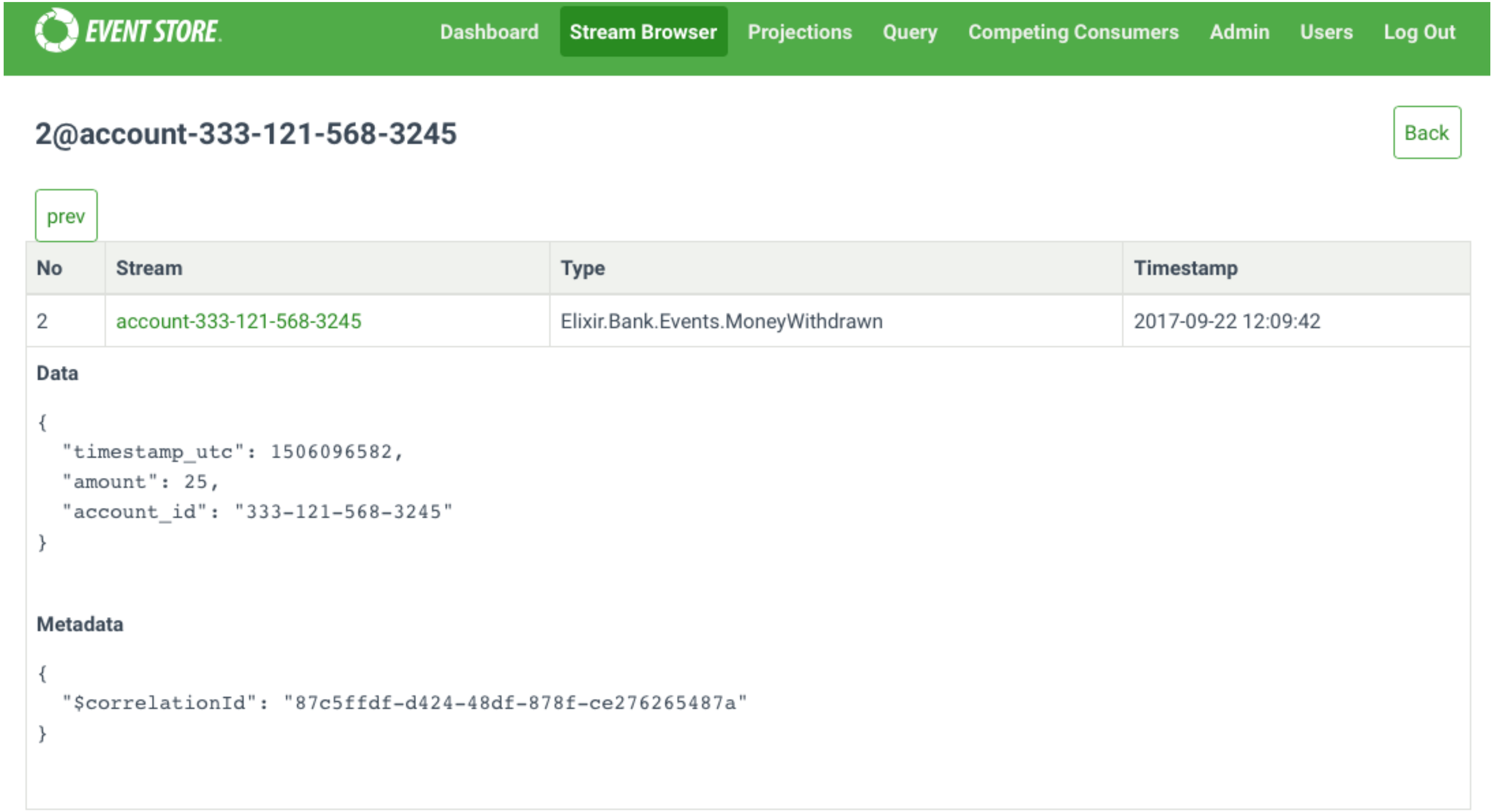
```
:ok
```



2@account-333-121-568-3245 Elixir.Bank.Events.MoneyWithdrawn

Event #	Name	Type	Created Date	
2	2@account-333-121-568-3245	Elixir.Bank.Events.MoneyWithdrawn	2017-09-22 12:09:42	JSON
1	1@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:08:32	JSON
0	0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59	JSON

Event Store 4.0.0.4 · Documentation · Support



2@account-333-121-568-3245

No	Stream	Type	Timestamp
2	account-333-121-568-3245	Elixir.Bank.Events.MoneyWithdrawn	2017-09-22 12:09:42

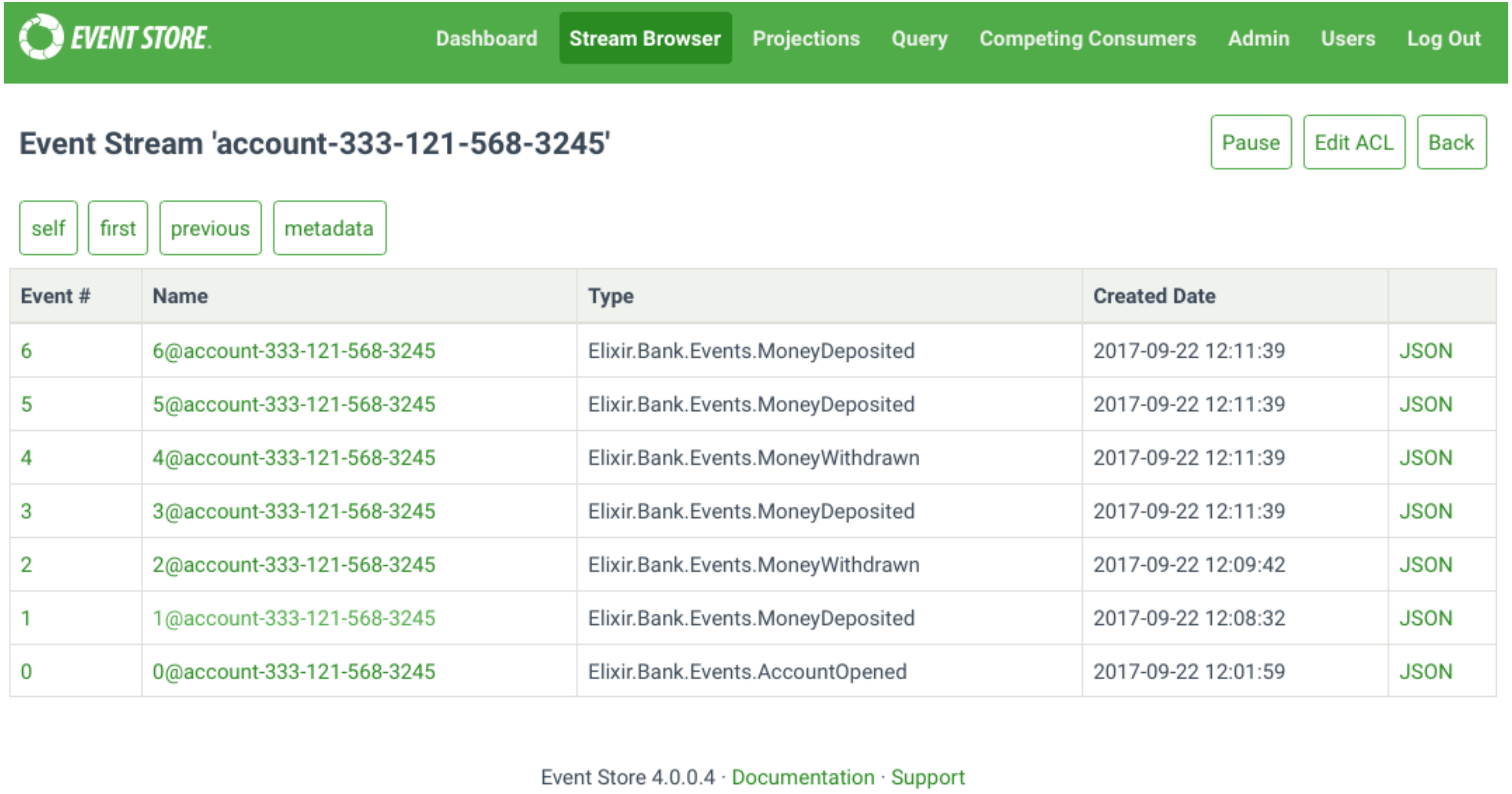
Data

```
{
  "timestamp_utc": 1506096582,
  "amount": 25,
  "account_id": "333-121-568-3245"
}
```

Metadata

```
{
  "$correlationId": "87c5ffdf-d424-48df-878f-ce276265487a"
}
```

Deposit $33
Withdraw $661
Deposit $500
Deposit $22



Event Stream 'account-333-121-568-3245'

Event #	Name	Type	Created Date	
6	6@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:11:39	JSON
5	5@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:11:39	JSON
4	4@account-333-121-568-3245	Elixir.Bank.Events.MoneyWithdrawn	2017-09-22 12:11:39	JSON
3	3@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:11:39	JSON
2	2@account-333-121-568-3245	Elixir.Bank.Events.MoneyWithdrawn	2017-09-22 12:09:42	JSON
1	1@account-333-121-568-3245	Elixir.Bank.Events.MoneyDeposited	2017-09-22 12:08:32	JSON
0	0@account-333-121-568-3245	Elixir.Bank.Events.AccountOpened	2017-09-22 12:01:59	JSON

Event Store 4.0.0.4 · Documentation · Support

Balance?

Balance?	Event
	Account Opened (initial)

Balance?	**Event**
balance = initial	Account Opened (initial)

Balance?

balance = initial

Event

Account Opened (initial)

Money Deposited (amount)

Balance?	**Event**
balance = initial	Account Opened (initial)
balance = balance + amount	Money Deposited (amount)

Balance?

balance = initial

balance = balance + amount

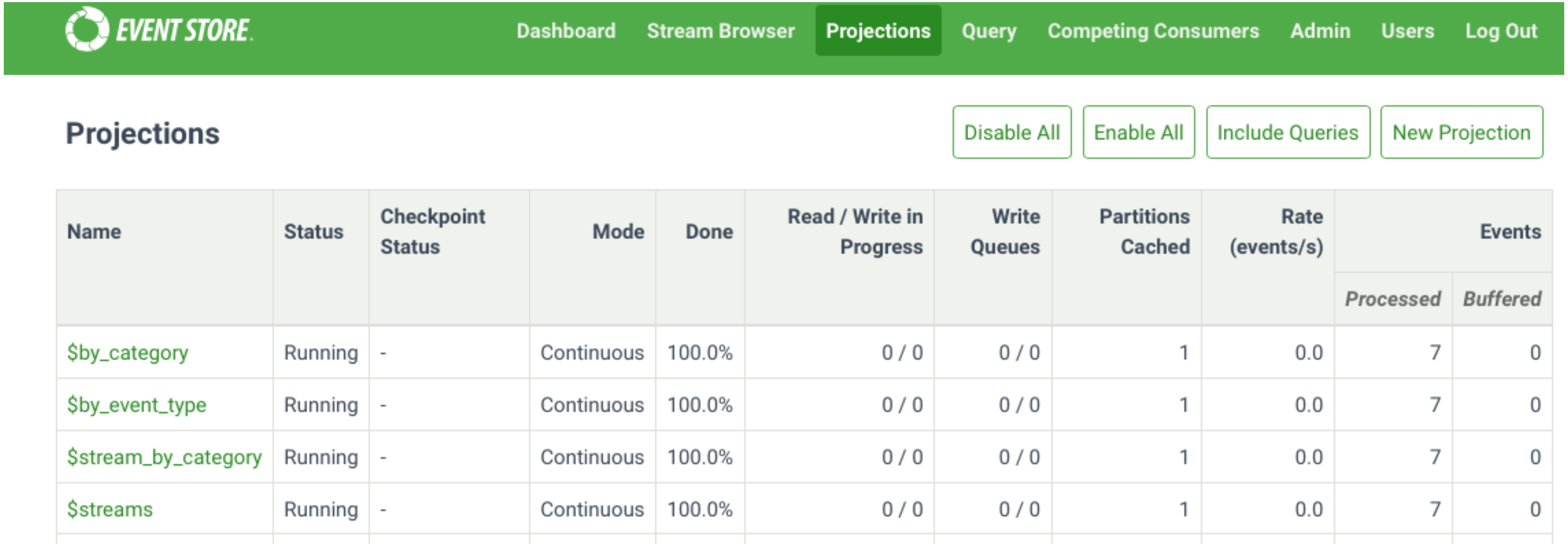
Event

Account Opened (initial)

Money Deposited (amount)

Money Withdrawn (amount)

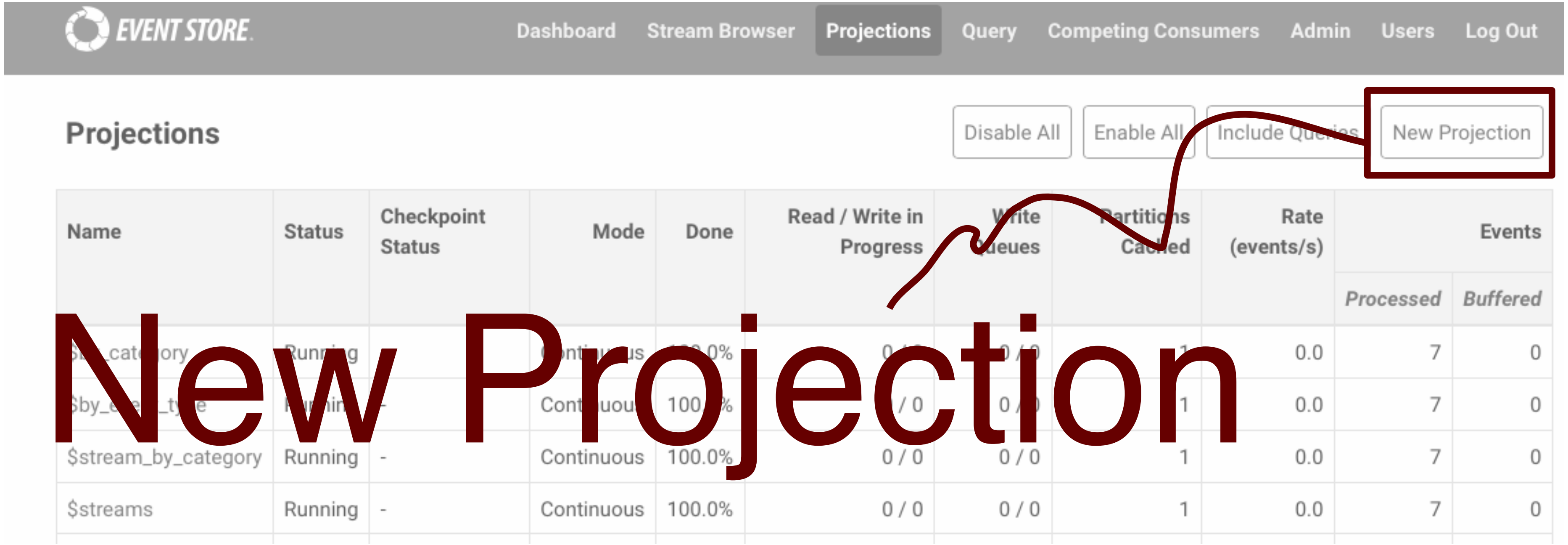
Balance?	Event
balance = initial	Account Opened (initial)
balance = balance + amount	Money Deposited (amount)
balance = balance - amount	Money Withdrawn (amount)



Projections

Name	Status	Checkpoint Status	Mode	Done	Read / Write in Progress	Write Queues	Partitions Cached	Rate (events/s)	Events	
									Processed	*Buffered*
$by_category	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$by_event_type	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$stream_by_category	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$streams	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0

New Projection



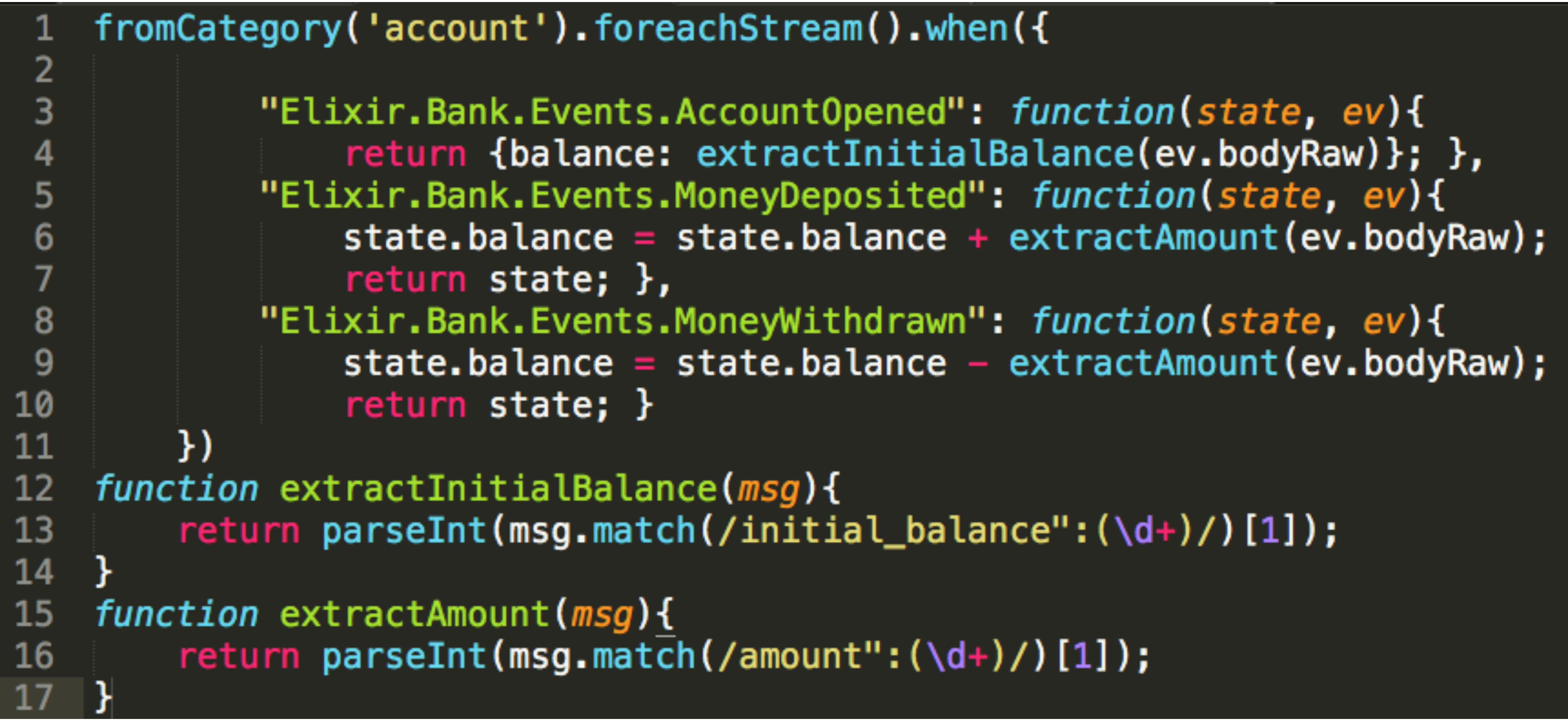
EVENT STORE

Dashboard | Stream Browser | Projections | Query | Competing Consumers | Admin | Users | Log Out

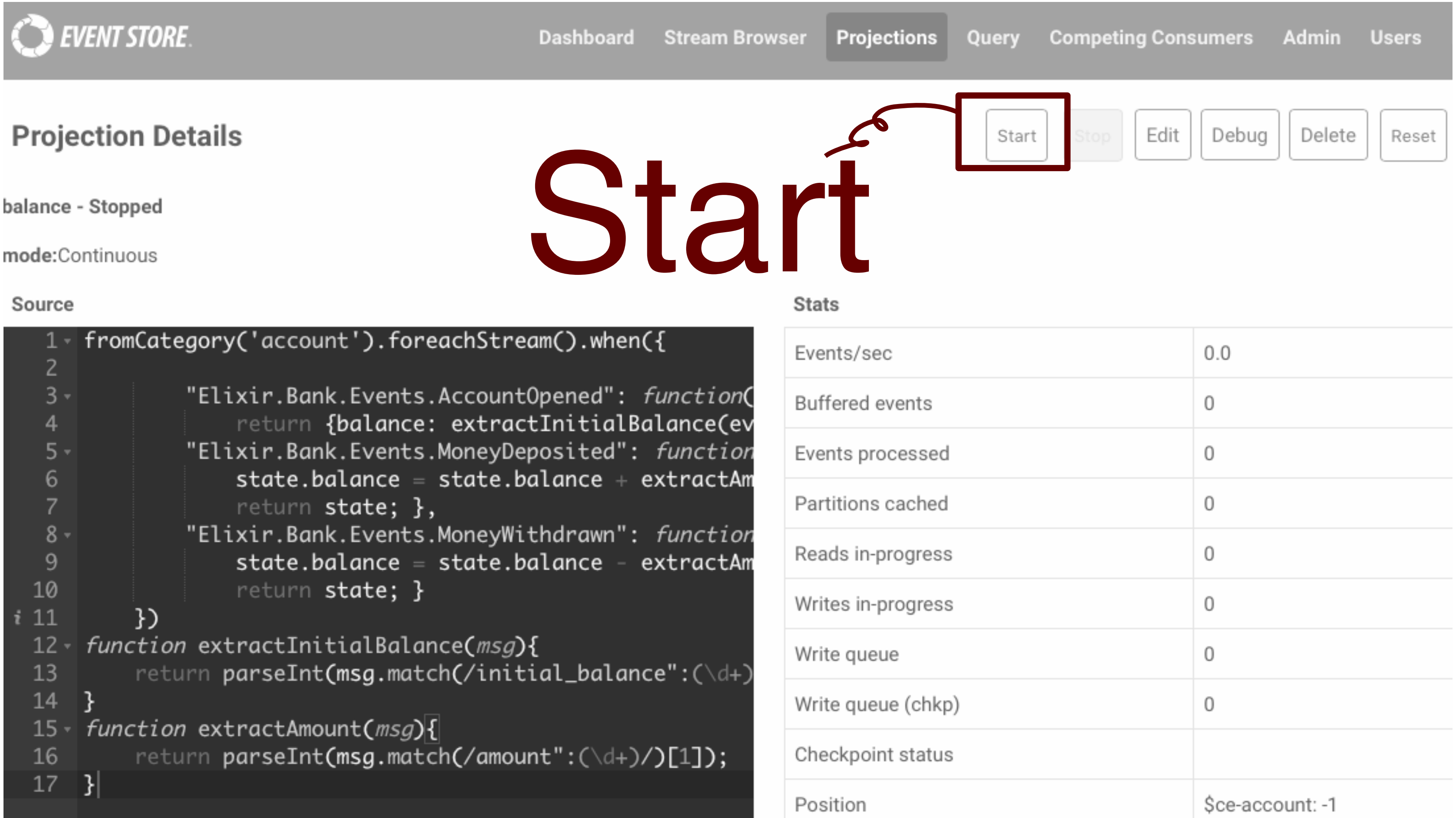
Projections

Disable All | Enable All | Include Queries | New Projection

Name	Status	Checkpoint Status	Mode	Done	Read / Write in Progress	Write Queues	Partitions Cached	Rate (events/s)	Events	
									Processed	*Buffered*
$by_category	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$by_event_type	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$stream_by_category	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0
$streams	Running	-	Continuous	100.0%	0 / 0	0 / 0	1	0.0	7	0



```javascript
fromCategory('account').foreachStream().when({

        "Elixir.Bank.Events.AccountOpened": function(state, ev){
            return {balance: extractInitialBalance(ev.bodyRaw)}; },
        "Elixir.Bank.Events.MoneyDeposited": function(state, ev){
            state.balance = state.balance + extractAmount(ev.bodyRaw);
            return state; },
        "Elixir.Bank.Events.MoneyWithdrawn": function(state, ev){
            state.balance = state.balance - extractAmount(ev.bodyRaw);
            return state; }
    })
function extractInitialBalance(msg){
    return parseInt(msg.match(/initial_balance":(\d+)/)[1]);
}
function extractAmount(msg){
    return parseInt(msg.match(/amount":(\d+)/)[1]);
}
```



Start

Projection Details

balance - Stopped

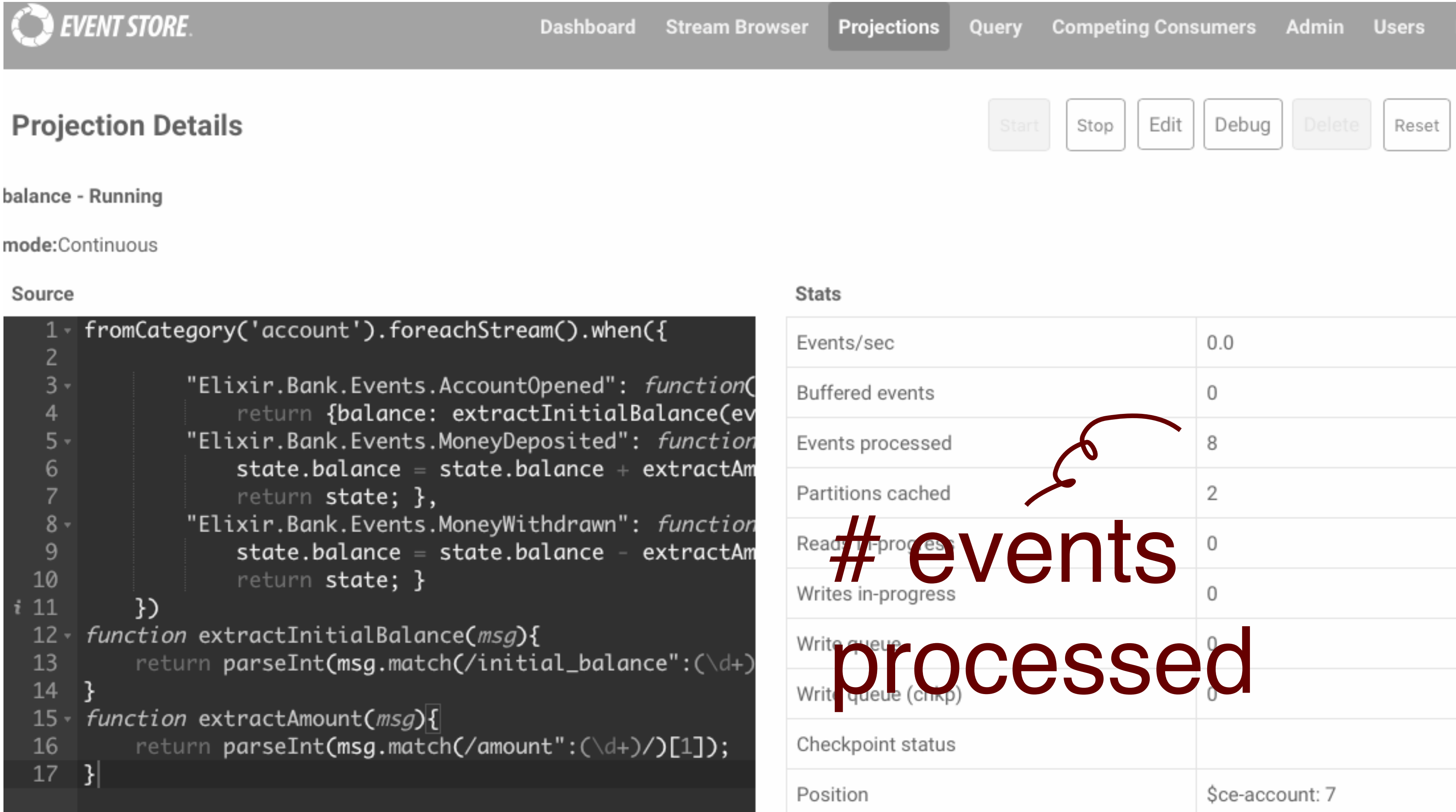
mode: Continuous

Source

```
fromCategory('account').foreachStream().when({

        "Elixir.Bank.Events.AccountOpened": function(
            return {balance: extractInitialBalance(ev
        "Elixir.Bank.Events.MoneyDeposited": function
            state.balance = state.balance + extractAm
            return state; },
        "Elixir.Bank.Events.MoneyWithdrawn": function
            state.balance = state.balance - extractAm
            return state; }
    })
function extractInitialBalance(msg){
    return parseInt(msg.match(/initial_balance":(\d+)
}
function extractAmount(msg){
    return parseInt(msg.match(/amount":(\d+)/)[1]);
}
```

Stats

Events/sec	0.0
Buffered events	0
Events processed	0
Partitions cached	0
Reads in-progress	0
Writes in-progress	0
Write queue	0
Write queue (chkp)	0
Checkpoint status	
Position	$ce-account: -1



Projection Details

balance - Running

mode:Continuous

Source

```
fromCategory('account').foreachStream().when({

        "Elixir.Bank.Events.AccountOpened": function(
            return {balance: extractInitialBalance(ev
        "Elixir.Bank.Events.MoneyDeposited": function
            state.balance = state.balance + extractAm
            return state; },
        "Elixir.Bank.Events.MoneyWithdrawn": function
            state.balance = state.balance - extractAm
            return state; }
    })
function extractInitialBalance(msg){
    return parseInt(msg.match(/initial_balance":(\d+)
}
function extractAmount(msg){
    return parseInt(msg.match(/amount":(\d+)/)[1]);
}
```

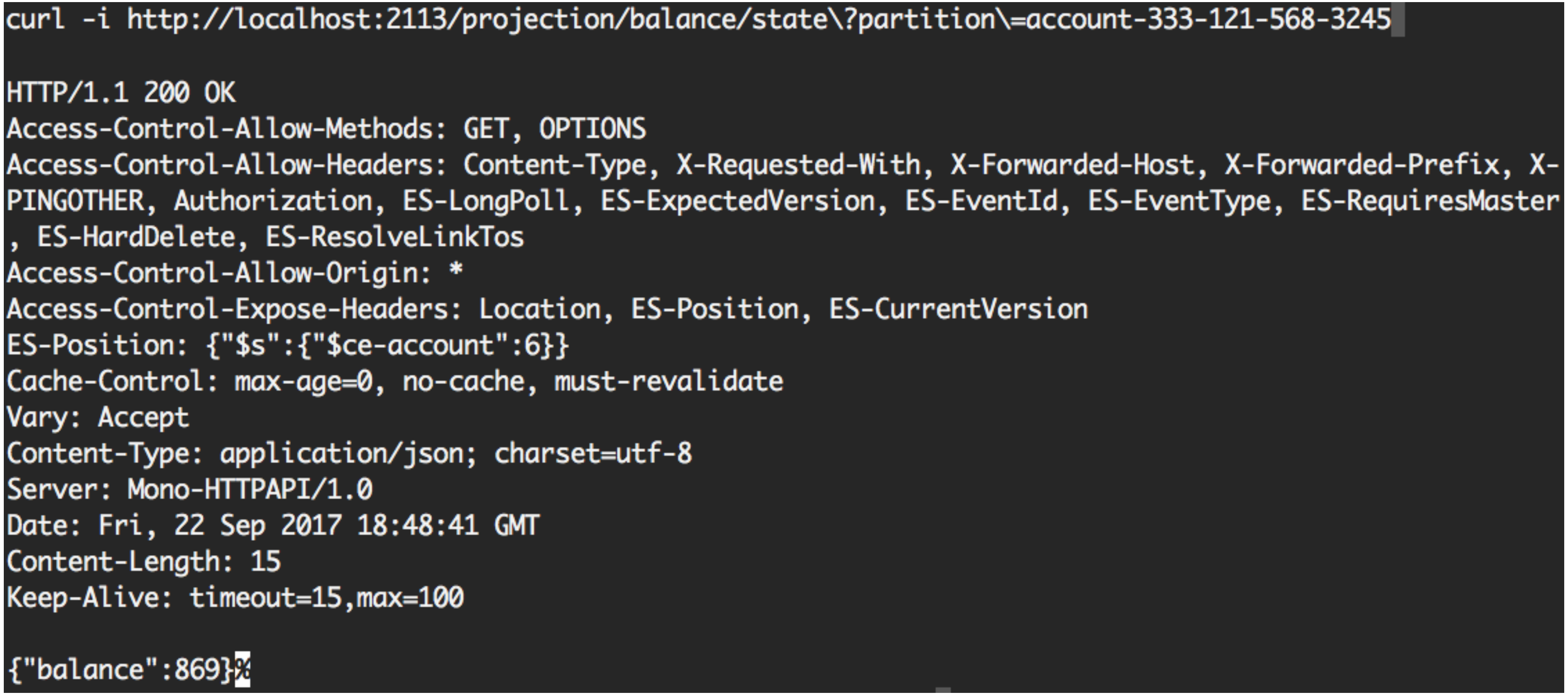
Stats

Events/sec	0.0
Buffered events	0
Events processed	8
Partitions cached	2
Reads in-progress	0
Writes in-progress	0
Write queue	0
Write queue (chkp)	0
Checkpoint status	
Position	$ce-account: 7

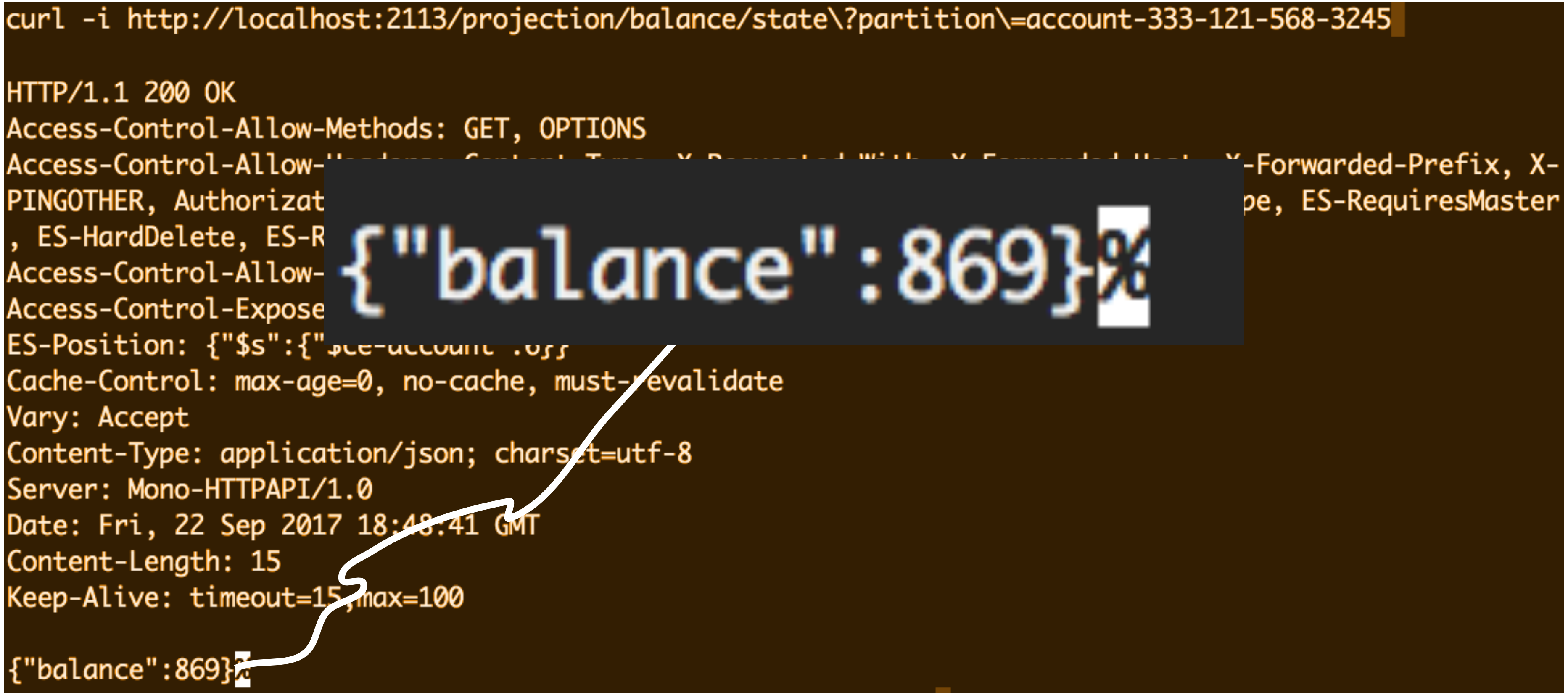
# events processed

GET request to:

projection/balance/state?partition=account-#



```
curl -i http://localhost:2113/projection/balance/state\?partition\=account-333-121-568-3245

HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET, OPTIONS
Access-Control-Allow-Headers: Content-Type, X-Requested-With, X-Forwarded-Host, X-Forwarded-Prefix, X-
PINGOTHER, Authorization, ES-LongPoll, ES-ExpectedVersion, ES-EventId, ES-EventType, ES-RequiresMaster
, ES-HardDelete, ES-ResolveLinkTos
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Location, ES-Position, ES-CurrentVersion
ES-Position: {"$s":{"$ce-account":6}}
Cache-Control: max-age=0, no-cache, must-revalidate
Vary: Accept
Content-Type: application/json; charset=utf-8
Server: Mono-HTTPAPI/1.0
Date: Fri, 22 Sep 2017 18:48:41 GMT
Content-Length: 15
Keep-Alive: timeout=15,max=100

{"balance":869}%
```



```
curl -i http://localhost:2113/projection/balance/state\?partition\=account-333-121-568-3245

HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET, OPTIONS
Access-Control-Allow-                                                    -Forwarded-Prefix, X-
PINGOTHER, Authorizat                                                    pe, ES-RequiresMaster
, ES-HardDelete, ES-R
Access-Control-Allow-
Access-Control-Expose
ES-Position: {"$s":{"$ce-account":6}}
Cache-Control: max-age=0, no-cache, must-revalidate
Vary: Accept
Content-Type: application/json; charset=utf-8
Server: Mono-HTTPAPI/1.0
Date: Fri, 22 Sep 2017 18:48:41 GMT
Content-Length: 15
Keep-Alive: timeout=15,max=100

{"balance":869}%
```

{"balance":869}%

</Demo>

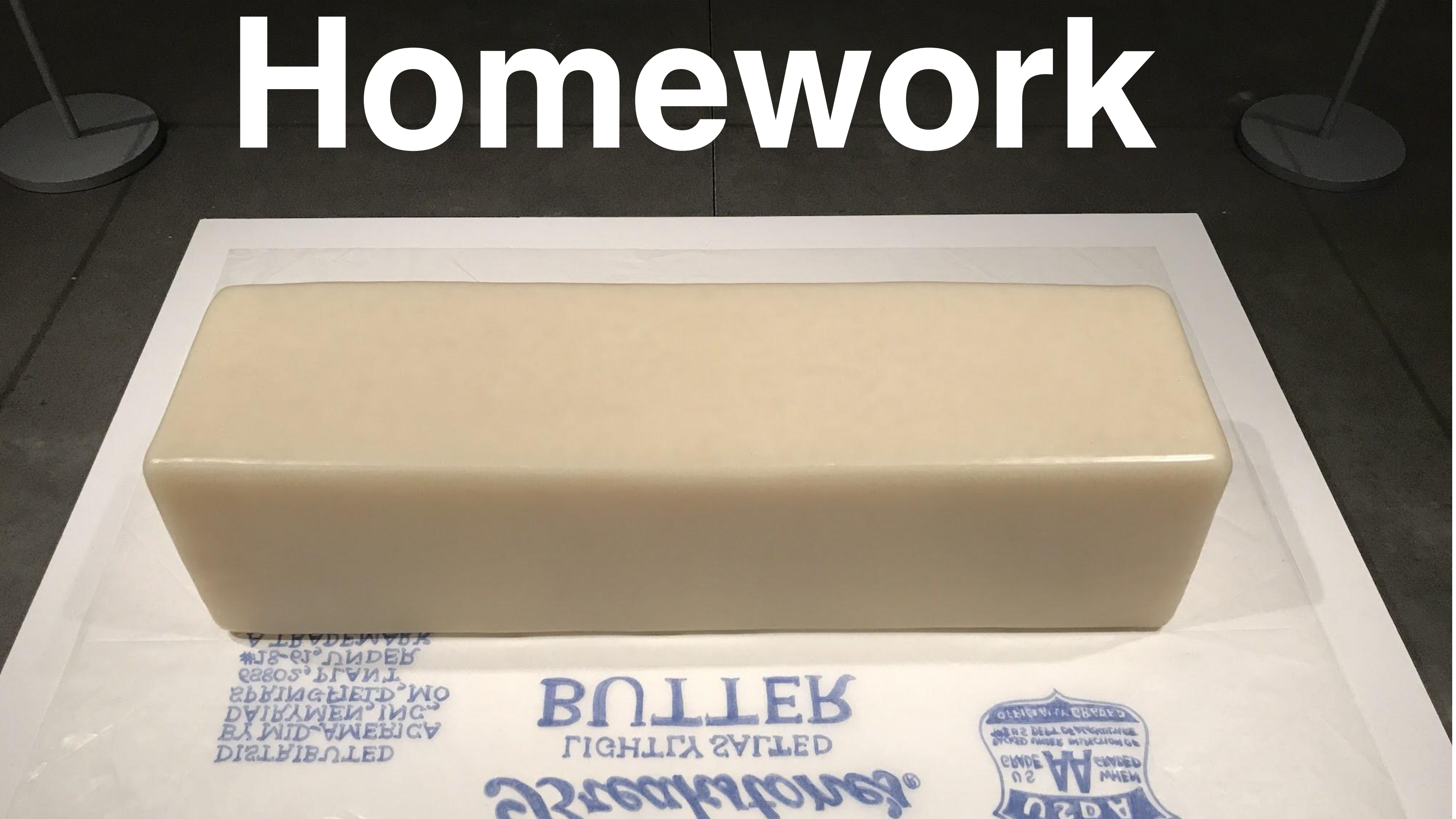
Issues

Commanded rough edges

Projections (Issue #74)	Cannot subscribe per-aggregate. No support for catch-up subscriptions.
Process Managers	Same as projections.
JSON body (Issue #4)	Stores content as escaped string.

Questions ?

Homework


BUTTER
LIGHTLY SALTED
DISTRIBUTED BY MID-AMERICA DAIRYMEN, INC. SPRINGFIELD, MO 65802, PLANT #29-61, UNDER A TRADEMARK
US GRADE AA WHEN GRADED
USDA

More commands

Overdraft protection?

Close account?

More Errors

Account does not exist for all commands

The end