Customer retention

and how to avoid double billing

Bryan Hunt (ESL)

About me

- Slinging code for 20 years. [brag_list]
- Writing Elixir for about 5 years now (and loving it).
- Built 52 slides so we'll have to move fast.

Erlang SOLUTIONS

www.erlang-solutions.com

[brag_list] Perl, VB, C, C++, PHP, Python, Java, Scala, Javascript, Actionscript, Erlang, Shell, Ansible, Zsh, AWK, Sed, etc, 😴

Why I like Elixir

- Elixir has lovely syntax and UX 
- Elixir makes hard things easy ✅
- Elixir also makes easy things easy ✅
- Elixir lacks drama ✅
- Elixir is stable/boring ✅
- Elixir doesn't crash 〽 [stg]

[stg] Unlike £

Talk objectives

1. Opportunity to complain about being double charged when booking a flight.
2. An analysis of why their system double charged the customer.
3. Illustrate techniques to:
   - Build a more reliable system
   - Not double charge customers
   - Proactively monitor for system malfunction

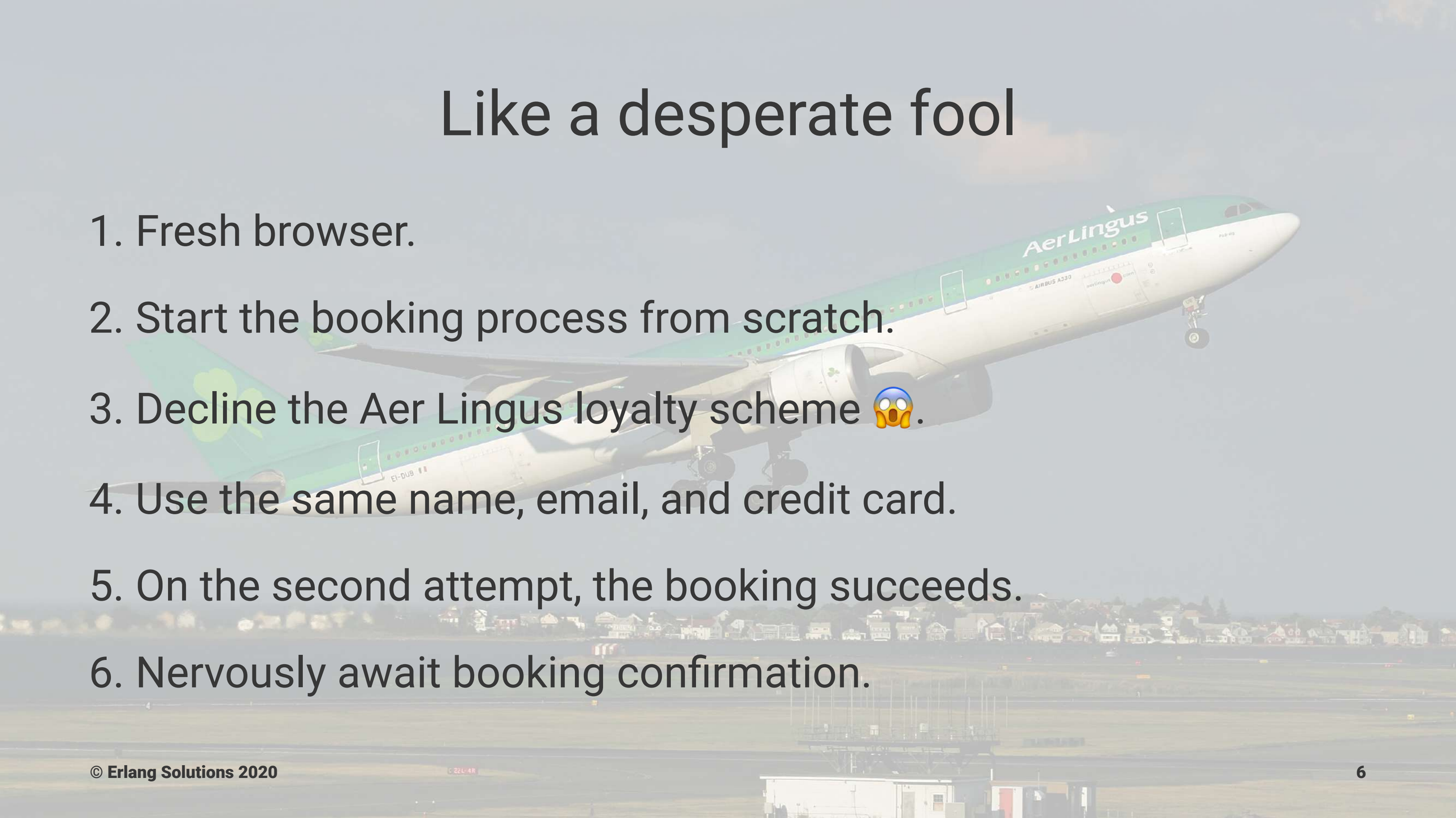
Opening scene [u owe me]

1. Urgently need to fly from Dublin to London the next day.
2. The website warns me there are only 3 seats remaining for the flight.
3. Better book fast!
   - Select flight.
   - Select any seat.
   - Enter my credit card details.
   - Deal with a couple of session timeouts.

1. The Aer Lingus website:
   - Insists I sign up for their loyalty scheme (the sweet, sweet irony).
   - Crashes.
   - Locks me out.
2. I wait 30 minutes and don't receive any email confirmation.
3. Panic 😧.

[u owe me] This happened nearly 4 years ago but I still want my £95.99 back (with interest).

Like a desperate fool



1. Fresh browser.
2. Start the booking process from scratch.
3. Decline the Aer Lingus loyalty scheme 😱.
4. Use the same name, email, and credit card.
5. On the second attempt, the booking succeeds.
6. Nervously await booking confirmation.

Success !

Receive a booking confirmation at 8:03 PM - you will be flying to London!

Actually Fail!

- Another booking confirmation, this time at 8:15 PM..
- Check my bank account - billed twice.
- Activate the online chat - it times out
- Call the website helpdesk... no answer..
- Maybe I'm not the only one having issues

Good luck with that !

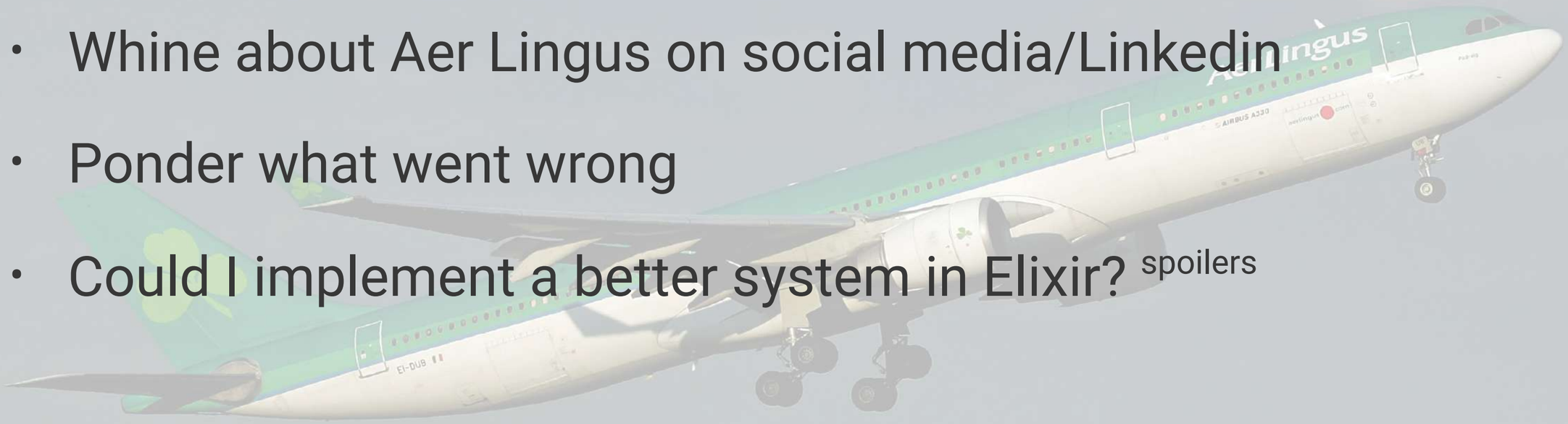
I can guarantee there's nobody listening on either of these two numbers

— Me

Website Helpdesk
0333 006 6920
Mon-Fri
08:00-06:00
Sat-Sun & Bank Holidays:
09:00-06:00

Reservations
0333 004 5000
Mon-Fri:
08:00-6:00
Sat-Sun & Bank Holidays:
09:00-06:00

Aftermath

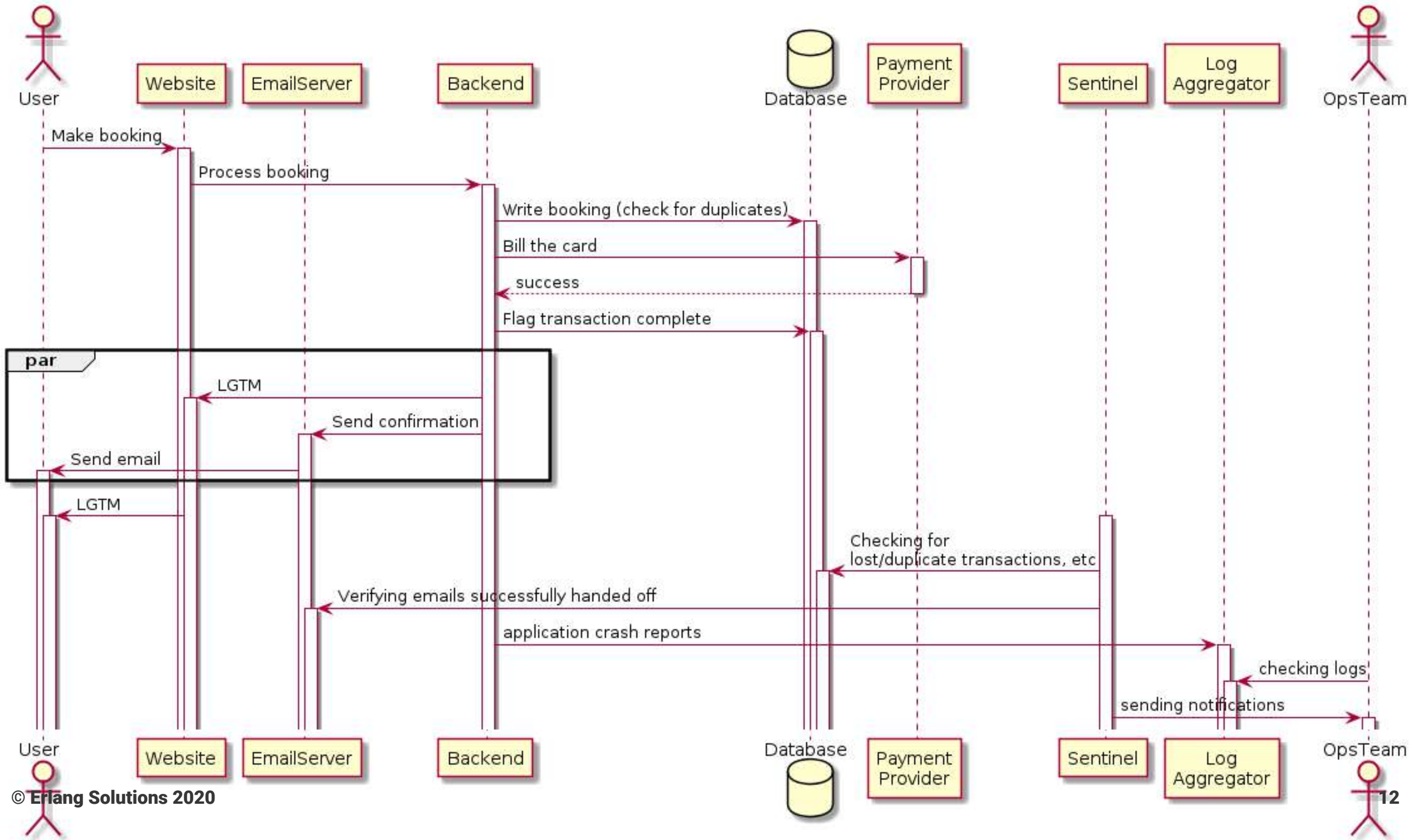


- Whine about Aer Lingus on social media/Linkedin
- Ponder what went wrong
- Could I implement a better system in Elixir? spoilers

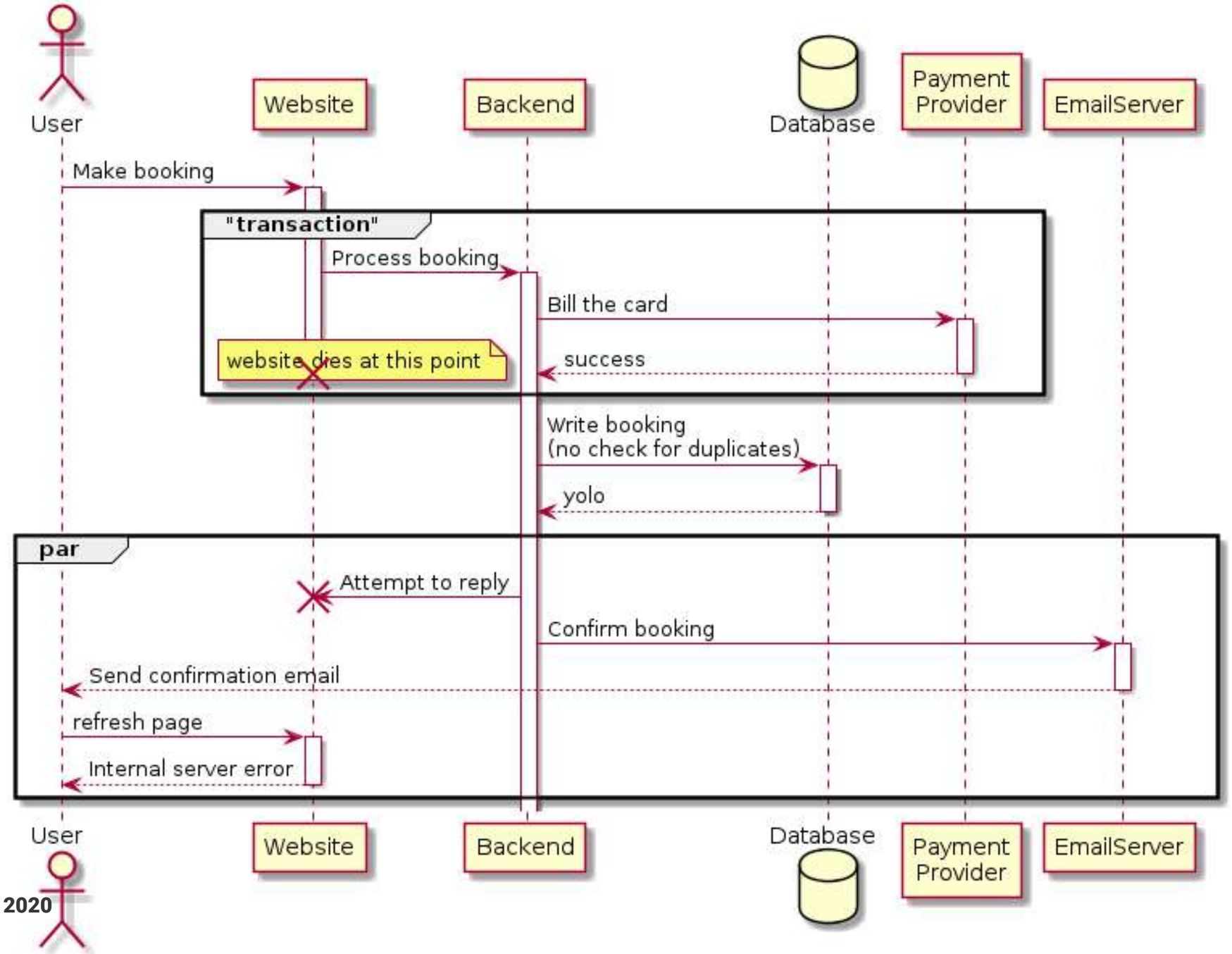
spoilers HTTP header leakage reveals it's running on Java/JBOSS - so of course we can..

What could we improve?

1. Error detection/notification
2. Fault tolerance
3. Session storage
4. Prevent double billing


How an ideal booking system would work
User
Website
EmailServer
Backend
Database
Payment Provider
Sentinel
Log Aggregator
OpsTeam
Make booking
Process booking
Write booking (check for duplicates)
Bill the card
success
Flag transaction complete
par
LGTM
Send confirmation
Send email
LGTM
Checking for lost/duplicate transactions, etc
Verifying emails successfully handed off
application crash reports
checking logs
sending notifications

How I suspect the Aer Lingus booking system is implemented



Error detection/notification

- Add bugsnag-elixir as a dependency and sign up for the bugsnag software as a service (GDPR?).
- Connect WombatOAM to the node [disclosure] and use it to detect errors.
- Write your own global error handler...

[disclosure] ESL product

Global error handler implementation

```elixir
defmodule Global.Handler do
  require Logger
  @behaviour :gen_event

  def init([]), do: {:ok, []}

  def handle_call({:configure, new_keys}, _state) do
    {:ok, :ok, new_keys}
  end

  def handle_event({:error_report, gl, {_pid, _type, [message | _]}}, state)
      when is_list(message) and node(gl) == node() do
      Logger.error("Global error handler: #{inspect(message, pretty: true)}")
      #Or maybe use the new (elixir 1.10.1) structured logger?
    {:ok, state}
  end

  def handle_event({_level, _gl, _event}, state) do
    {:ok, state}
  end
end
```

Using the global error handler

Add it :

```
:error_logger.add_report_handler(Global.Handler)
```

Trap exit signals

```
Process.flag(:trap_exit,true)
```

Spawn at process which will raise an exception/terminate

```
Task.async(fn -> raise "hell" end)
```

```
nil
iex(12)>
nil
iex(13)>
nil
iex(14)>
nil
iex(15)>
nil
iex(16)>
nil
iex(17)>
nil
iex(18)> defmodule Global.Logger do
...(18)>   require Logger
...(18)>   @behaviour :gen_event
...(18)>
...(18)>   def init([]), do: {:ok, []}
...(18)>
...(18)>   def handle_call({:configure, new_keys}, _state) do
...(18)>     {:ok, :ok, new_keys}
...(18)>   end
...(18)>
...(18)>   def handle_event({:error_report, gl, {_pid, _type, [message | _]}}, state)
...(18)>       when is_list(message) and node(gl) == node() do
...(18)>       Logger.error("Global error handler: #{inspect(message, pretty: true)}")
...(18)>     {:ok, state}
...(18)>   end
...(18)>
...(18)>   def handle_event({_level, _gl, _event}, state) do
...(18)>     {:ok, state}
...(18)>   end
...(18)> end
{:module, Global.Logger,
 <<70, 79, 82, 49, 0, 0, 8, 232, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 1, 14,
   0, 0, 0, 27, 20, 69, 108, 105, 120, 105, 114, 46, 71, 108, 111, 98, 97, 108,
   46, 76, 111, 103, 103, 101, 114, 8, 95, ...>>, {:handle_event, 2}}
iex(19)>
```

Fault tolerance

Lets concentrate on handling calls to another system which is temporarily failing.

We need an intelligent way to retry failed operations

- Code your own retry handling logic?
- Luke! Use the (open) source!

Options:

- safwank/ElixirRetry
- IanLuites/with_retry

Using Retry library (safwank/ElixirRetry)

```
use Retry
retry with: exponential_backoff(1000,2) |>
  jitter() |>
  Enum.take(5) do
  countdown = Process.get(:countdown,0)
  IO.puts("counter: #{countdown}, #{DateTime.utc_now}" )
  if countdown < 4 do
    Process.put(:countdown, countdown + 1)
    raise "countdown too low - trying again..."
  else
    :ok
  end
  after
    result -> result
  else
    error -> error
end
```

Quick shout out to the Elixir macro overlords [java (™)]

```
cat retry4j/src/**/*.java | wc -l
    3178

cat deps/retry/lib/**/*.ex | wc -l
     464
```

[java (™)] And I'm so grateful not to be coding Java...

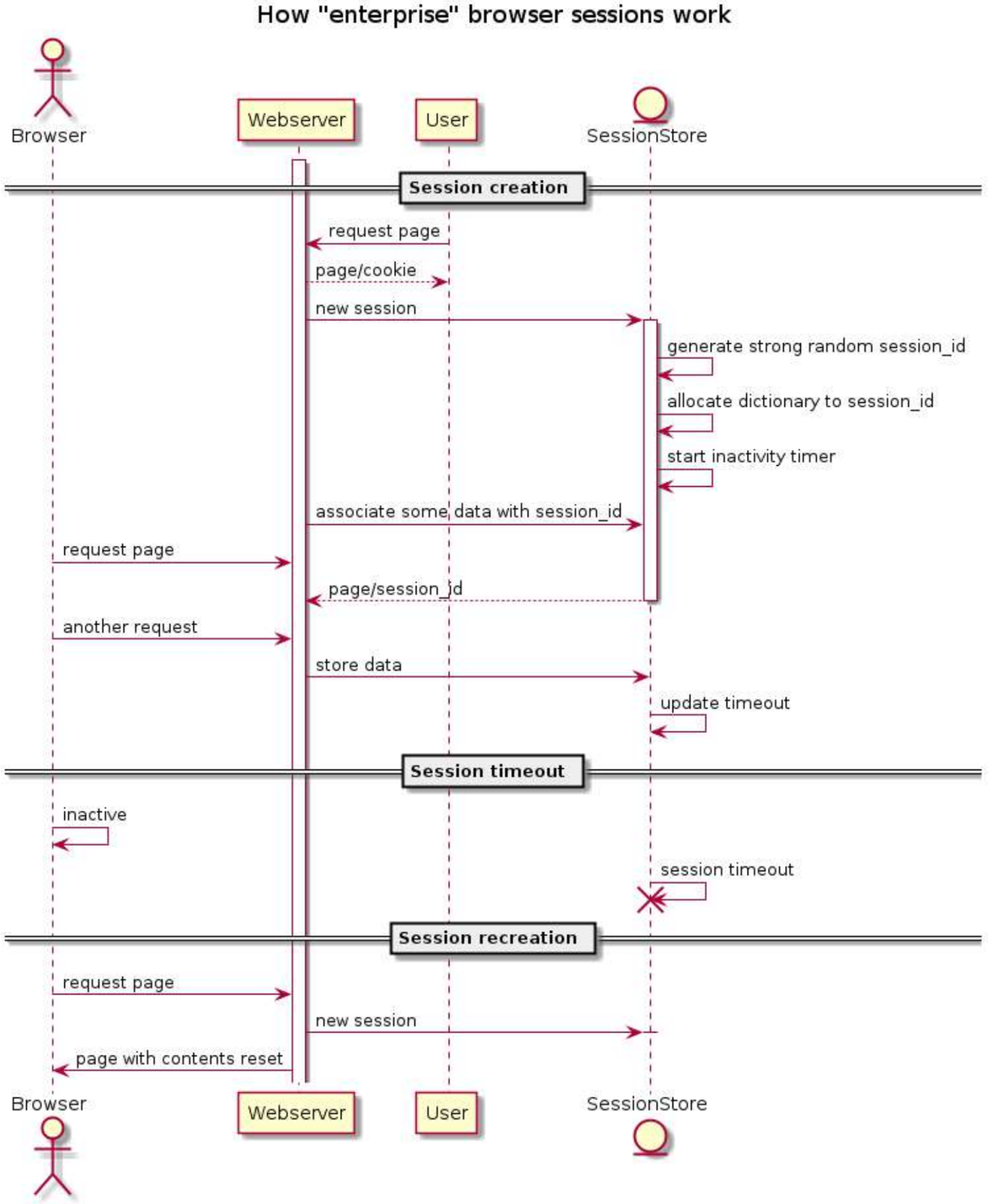
Session storage

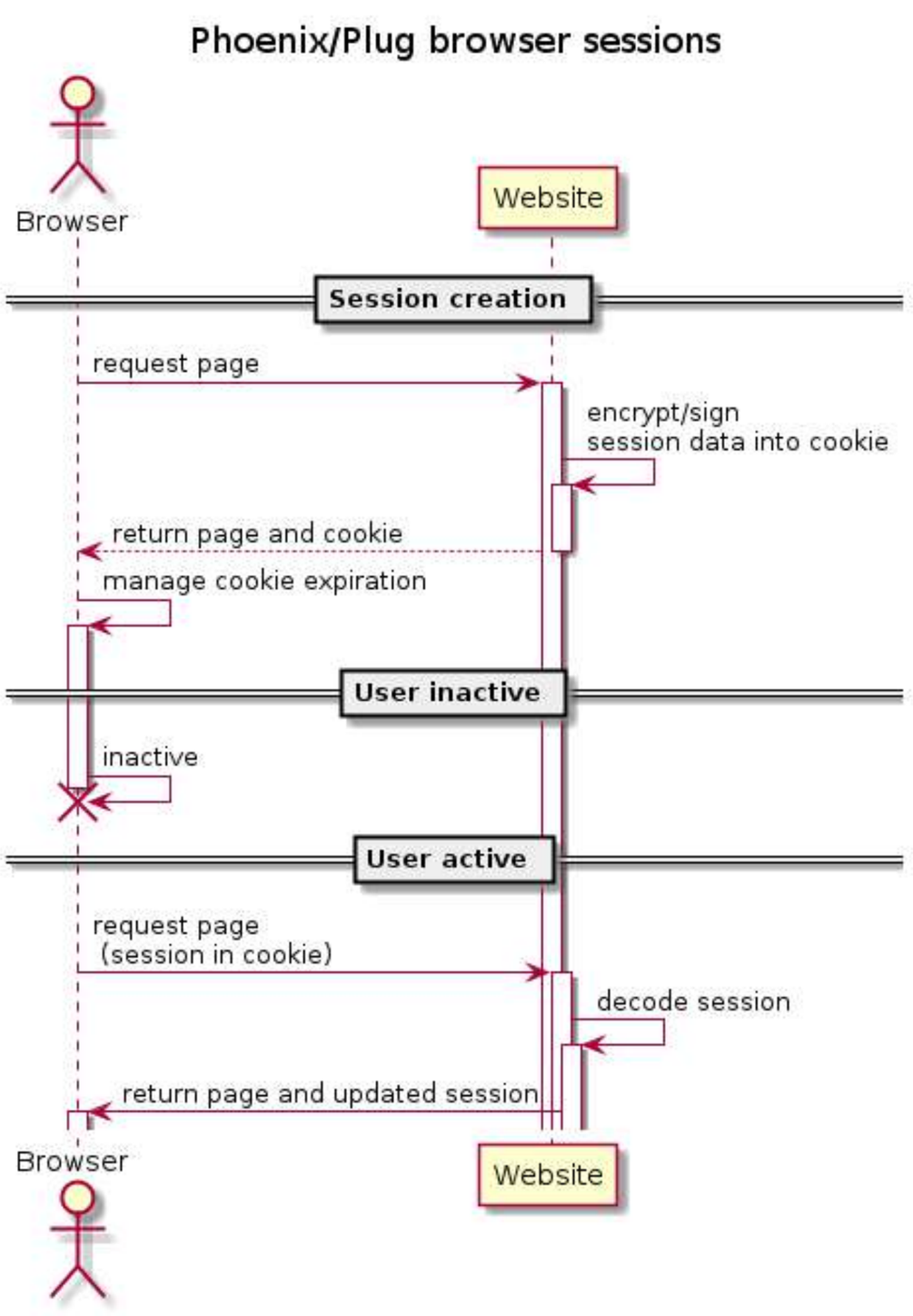
"Your session has timed out after 5 minutes of inactivity, please start again and wade through the 20 screens the marketing "people" insisted on adding to the booking flow..."

— *Every Java/.Net website ever*

^Ever notice how when the session times out - airline websites always set the travel dates to two weeks in the future - don't they know what cookies are for?

Session storage in Plug/Phoenix





How do we configure session storage in Phoenix/Plug

endpoint.ex

```
plug Plug.Session,
  store: :cookie,
  key: "_chat_key",
  signing_salt: "cKjB7sPT"
  max_age: 24*60*60*30  # 30 days
```

Trivial

Prevent double billing

Unique database constraints

Ecto (Elixir persistence framework)

Basic concepts

- Migration
- Changeset
- Repo
- Conveniences

Migration

```
defmodule Airline.Repo.Migrations.CreateFlightBookings do
  use Ecto.Migration

  def change do
    create table(:flight_bookings) do
      add :name, :string
      add :surname, :string
      add :cc_hash, :string
      add :flight_number, :string
      add :minute, :string
      add :hour, :string
      add :day, :string
      add :month, :string
      add :year, :string

      timestamps()
    end
.....
```

```
    create unique_index(
             :flight_bookings,
             [:name,
              :surname,
              :cc_hash,
              :flight_number,
              :minute,
              :hour,
              :day,
              :month,
              :year ],
             name: :unique_traveller_index)
  end
end
```

Changeset

```elixir
defmodule Airline.Flight.Booking do
  use Ecto.Schema
  import Ecto.Changeset

  schema "flight_bookings" do
    field :cc_hash, :string
    field :day, :string
    field :flight_number, :string
    field :hour, :string
    field :minute, :string
    field :month, :string
    field :name, :string
    field :surname, :string
    field :year, :string

    timestamps()
  end
```

```elixir
  @doc false
  def changeset(booking, %{} = attrs) do
    booking |> cast(attrs,[
                    :name,
                    :surname,
                    :cc_hash,
                    :flight_number,
                    :minute,
                    :hour,
                    :day,
                    :month,
                    :year ])
    |> validate_required([
                    :name,
                    :surname,
                    :cc_hash,
                    :flight_number,
                    :minute,
                    :hour,
                    :day,
                    :month,
                    :year])
    |>  unique_constraint(:unique_booking_constraint,
name: :unique_traveller_index)
  end

end
```

Repo

A repository maps to an underlying data store, controlled by the adapter. For example, Ecto ships with a Postgres adapter that stores data into a PostgreSQL database.

Convenience

Generated most of the prior code with the following command:

```
mix phx.gen.schema \
  Booking \
  flight_bookings \
  name \
  surname \
  cc_hash \
  flight_number \
  minute \
  hour \
  day \
  month \
  year
```

Use the Ecto changeset to validate input without touching the database

```
iex(8)> Airline.Flight.Booking.changeset(%Airline.Flight.Booking{}, %{})
#Ecto.Changeset<
  action: nil,
  changes: %{},
  errors: [
    name: {"can't be blank", [validation: :required]},
    surname: {"can't be blank", [validation: :required]},
    cc_hash: {"can't be blank", [validation: :required]},
    pp_hash: {"can't be blank", [validation: :required]},
    flight_number: {"can't be blank", [validation: :required]},
    minute: {"can't be blank", [validation: :required]},
    hour: {"can't be blank", [validation: :required]},
    day: {"can't be blank", [validation: :required]},
    month: {"can't be blank", [validation: :required]},
    year: {"can't be blank", [validation: :required]}
  ],
  data: #Airline.Flight.Booking<>,
  valid?: false
>
```

Generate a valid changeset

```
cc_num_hash = :crypto.hash(:sha256,"5105105105105100") |> Base.encode64


input = %{
  name: "davey",
  surname: "jones",
  cc_hash: cc_num_hash,
  flight_number: "flight_number",
  minute: "minute",
  hour: "hour",
  day: "day",
  month: "month",
  year: "year"
}

valid_changeset = %Ecto.Changeset{valid?: true} = Airline.Flight.Booking.changeset(%Airline.Flight.Booking{}, input)
```

Insert valid data

```
iex(7)> Airline.Repo.insert(valid_changeset)
[debug] QUERY OK db=3.4ms decode=1.4ms queue=2.2ms idle=9906.6ms
INSERT INTO "flight_bookings" ("cc_hash","day", SNIP...
{:ok,
 %Airline.Flight.Booking{
   __meta__: #Ecto.Schema.Metadata<:loaded, "flight_bookings">,
   cc_hash: "cc_hash",
   day: "day",
   flight_number: "flight_number",
   hour: "hour",
   id: 1,
   inserted_at: ~N[2020-02-28 22:20:54],
   minute: "minute",
   month: "month",
   name: "name",
   surname: "surname",
   updated_at: ~N[2020-02-28 22:20:54],
   year: "year"
 }}
```

Insert duplicate data

```
iex(8)> Airline.Repo.insert(valid_changeset)

[debug] QUERY ERROR db=7.4ms queue=1.9ms idle=9324.1ms
INSERT INTO "flight_bookings" ("cc_hash","day", SNIP...
{:error,
 #Ecto.Changeset<
   action: :insert,
   changes: %{
     cc_hash: "cc_hash",
     day: "day",
     flight_number: "flight_number",
     hour: "hour",
     minute: "minute",
     month: "month",
     name: "name",
     surname: "surname",
     year: "year"
   },
   errors: [
     unique_booking_constraint: {"has already been taken", [constraint: :unique, constraint_name: "unique_traveller_index"]}
   ],
   data: #Airline.Flight.Booking<>,
   valid?: false
 >}
```

Lets try something a little more efficient

We add an `:entity_hash` column to the Booking module

```elixir
defmodule Airline.Flight.Booking do
  use Ecto.Schema
  import Ecto.Changeset

  @required_attrs [ :name, :surname, :cc_hash, :entity_hash, :flight_number, :minute, :hour, :day, :month, :year ]

  @hash_attrs @required_attrs

  schema "flight_bookings" do
    field :cc_hash, :string
    field :entity_hash, :string
    field :day, :string
    field :flight_number, :string
    field :hour, :string
    field :minute, :string
    field :month, :string
    field :name, :string
    field :surname, :string
    field :year, :string

    timestamps()
  end
```

And modify the changeset function to calculate the hash of the unique fields before we store a booking to the database

```
  @doc false
  def changeset(booking, %{} = attrs) do
    entity_hash =
      :crypto.hash(:sha256, inspect(Map.to_list(attrs |> Map.take(@hash_attrs))))
      |> Base.encode64()

    augmented_attrs = Map.put(attrs, :entity_hash, entity_hash)

    booking
    |> cast(
      augmented_attrs,
      @required_attrs
    )
    |> validate_required(@required_attrs)
    |> unique_constraint(:unique_booking_constraint, name: :unique_traveller_index)
  end
end
```

The schema/migration now becomes the much more reasonable

```elixir
defmodule Airline.Repo.Migrations.CreateFlightBookings do
  use Ecto.Migration

  def change do
    create table(:flight_bookings) do
      add :name, :string
      add :surname, :string
      add :cc_hash, :string
      add :entity_hash, :string
      add :flight_number, :string
      add :minute, :string
      add :hour, :string
      add :day, :string
      add :month, :string
      add :year, :string
      timestamps()
    end

    create unique_index( :flight_bookings, [ :entity_hash ], name: :unique_traveller_index)
  end
end
```

```
iex(17)>
nil
iex(18)>
nil
iex(19)>
nil
iex(20)>
nil
iex(21)>
nil
iex(22)>
nil
iex(23)>
nil
iex(24)>
nil
iex(25)>
nil
iex(26)>
nil
iex(27)>
nil
iex(28)>
nil
iex(29)> input = %{
...(29)>   name: "davey",
...(29)>   surname: "jones",
...(29)>   cc_hash: "MElF6R3j3v9Sph0IczFB1y3ULsnUeXLxBgU01UwMf5A=",
...(29)>   flight_number: "flight_number",
...(29)>   minute: "minute",
...(29)>   hour: "hour",
...(29)>   day: "day",
...(29)>   month: "month",
...(29)>   year: "year"
...(29)> }
```

Can we be even more efficient?

Can we know if a booking has already passed through the system without touching the database?

Bloom filter [bloom]

Used as an optimization in many data stores to avoid hitting index files to check if an element exists - some examples :

- Cassandra
- Riak

[bloom] A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set https://en.wikipedia.org/wiki/Bloom_filter

Using a bloom filter (gmcabrita/bloomex)

```elixir
defmodule Bloomer do
 use GenServer

  def start_link(_) do
    GenServer.start_link(__MODULE__, nil, name: __MODULE__)
  end

  def add(element) do
    GenServer.cast( __MODULE__, {:add, element})
  end

  def exists(element) do
    GenServer.call( __MODULE__, {:exists, element})
  end

  @impl true
  def init(_) do
    {:ok, Bloomex.scalable(1000, 0.1, 0.1, 2) }
  end

  @impl true
  def handle_call({:exists,element} , _from, state) do
    exists = Bloomex.member?(state, element)
    {:reply, exists, state}
  end

  @impl true
  def handle_cast({:add, element}, state) do
    {:noreply, Bloomex.add(state, element) }
  end
end
```

Add the GenServer to the supervision tree of your application module

```
defmodule Airline.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  def start(_type, _args) do
    # List all child processes to be supervised
    children = [
      Bloomer,
      Airline.Repo
```

```
b@krazyvui          Sat Feb 29 09:27:28P
~/repos/bhesl/chat/ iex -S mix
Erlang/OTP 22 [erts-10.5.6] [source] [64-bit] [smp:12:12] [ds:12:12:10] [async-threads:1] [hipe]

[debug] registry startup options = '[members: [:nonode@nohost], keys: :unique]
[info] Starting Horde.RegistryImpl with name Chat.PollerRegistry
Starting: Elixir.Chat.PollerSupervisor
[info] Starting Horde.DynamicSupervisorImpl with name Chat.PollerSupervisor
Interactive Elixir (1.9.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

Add the bloom filter into the storage module

```
changeset = Airline.Flight.Booking.changeset(%Airline.Flight.Booking{}, booking)
if Bloomer.exists {:booking, changeset.changes.entity_hash}  do
  Logger.warn("Possible duplicate booking #{inspect(booking)}")
end
Bloomer.add {:booking, changeset.changes.entity_hash}
```

What about the database (or other service) being temporarily unavailable ?

Remember Retry?

We can use Retry to retry database inserts

```
defmodule Bookings do

  import Ecto.Query, warn: false
  alias Airline.Repo
  alias Airline.Flight.Booking

  def insert_booking_with_retry( %{ name:_, surname:_, cc_hash:_, flight_number:_, minute:_, hour:_, day:_, month:_, year:_ } = booking) do
    use Retry

    retry with: exponential_backoff()  |> Enum.take(10) , rescue_only: [DBConnection.ConnectionError]   do
      IO.puts("attempting to insert changeset - #{DateTime.utc_now}")
      changeset = Airline.Flight.Booking.changeset(%Airline.Flight.Booking{}, booking)
      case Repo.insert(changeset) do
        {:error, changeset = %{valid?: false}  } -> {:invalid_changeset, changeset }
        other -> other
      end
    after
      result -> result
    else
      error -> error
    end
  end
end
```

```
nil
iex(9)>
nil
iex(10)>
nil
iex(11)>
nil
iex(12)>
nil
iex(13)>
```

demo

```
23:01:51 UTC
2020-02-29 23:05:15.870 UTC [1] LOG:  database system is ready to accept connection
s
```

What could we improve?

1. Error detection/notification ✅
2. Fault tolerance ✅
3. Session storage ✅
4. Prevent double billing ✅

Thanks for listening!

Slide (markdown) content can be found at

https://github.com/esl/bryan_cb_sf_2020_talk

Thank you to :
* Erlang team
* Elixir team
* The open source community
* Erlang solutions for flying me out to sunny USA