Swoosh

hex v1.6.3 | downloads 2.1M | license MIT | last commit february

Compose, deliver and test your emails easily in Elixir.

We have applied the lessons learned from projects like Plug, Ecto and Phoenix in designing clean and composable APIs, with clear separation of concerns between modules. Swoosh comes with 12 adapters, including SendGrid, Mandrill, Mailgun, Postmark and SMTP. See the full list of adapters below.

The complete documentation for Swoosh is available online at HexDocs.

Requirements

Elixir 1.10+ and Erlang OTP 22+

Getting started

```
# In your config/config.exs file
config :sample, Sample.Mailer,
  adapter: Swoosh.Adapters.Sendgrid,
  api_key: "SG.x.x"
```

```
# In your application code
defmodule Sample.Mailer do
  use Swoosh.Mailer, otp_app: :sample
end
```

```
defmodule Sample.UserEmail do
  import Swoosh.Email

  def welcome(user) do
    new()
    |> to({user.name, user.email})
    |> from({"Dr B Banner", "hulk.smash@example.com"})
    |> subject("Hello, Avengers!")
    |> html_body("<h1>Hello #{user.name}</h1>")
    |> text_body("Hello #{user.name}\n")
  end
end
```

```
# In an IEx session
email = Sample.UserEmail.welcome(%{name: "Tony Stark", email: "tony.stark@example.com"})
Sample.Mailer.deliver(email)
```

```
# Or in a Phoenix controller
defmodule Sample.UserController do
  use Phoenix.Controller
  alias Sample.UserEmail
  alias Sample.Mailer

  def create(conn, params) do
    user = create_user!(params)

    UserEmail.welcome(user) |> Mailer.deliver()
  end
end
```

See `Swoosh.Mailer` for more configuration options.

Installation

- Add swoosh to your list of dependencies in `mix.exs`:

  ```
  def deps do
    [{:swoosh, "~> 1.6"}]
  end
  ```

- (Optional-ish) Most adapters (non SMTP ones) use `Swoosh.ApiClient` to talk to the service provider. Swoosh comes with `Swoosh.ApiClient.Hackney` configured by default. If you want to use it, you just need to include `Hackney` as a dependency of your app.

  Swoosh also accepts `Finch` out-of-the-box. See `Swoosh.ApiClient.Finch` for details. If you need to integrate with another HTTP client, it's easy to define a new API client. Follow the `Swoosh.ApiClient` behaviour and configure Swoosh to use it:

  ```
  config :swoosh, :api_client, MyApp.ApiClient
  ```

  But if you don't need `Swoosh.ApiClient`, you can disable it by setting the value to `false`:

  ```
  config :swoosh, :api_client, false
  ```

  This is the case when you are using `Swoosh.Adapters.Local`, `Swoosh.Adapters.Test` and adapters that are SMTP based, that don't require an API client.

- (Optional) If you are using `Swoosh.Adapters.SMTP`, `Swoosh.Adapters.Sendmail` or `Swoosh.Adapters.AmazonSES`, you also need to add `gen_smtp` to your dependencies:

  ```
  def deps do
    [
      {:swoosh, "~> 1.6"},
      {:gen_smtp, "~> 1.0"}
    ]
  end
  ```

Adapters

Swoosh supports the most popular transactional email providers out of the box and also has a SMTP adapter. Below is the list of the adapters currently included:

PROVIDER	SWOOSH ADAPTER
SMTP	Swoosh.Adapters.SMTP
SendGrid	Swoosh.Adapters.Sendgrid
Sendinblue	Swoosh.Adapters.Sendinblue
Sendmail	Swoosh.Adapters.Sendmail
Mandrill	Swoosh.Adapters.Mandrill
Mailgun	Swoosh.Adapters.Mailgun
Mailjet	Swoosh.Adapters.Mailjet
Postmark	Swoosh.Adapters.Postmark
SparkPost	Swoosh.Adapters.SparkPost
Amazon SES	Swoosh.Adapters.AmazonSES
Dyn	Swoosh.Adapters.Dyn
SocketLabs	Swoosh.Adapters.SocketLabs
Gmail	Swoosh.Adapters.Gmail
MailPace	Swoosh.Adapters.MailPace

Configure which adapter you want to use by updating your `config/config.exs` file:

```
config :sample, Sample.Mailer,
  adapter: Swoosh.Adapters.SMTP
  # adapter config (api keys, etc.)
```

Check the documentation of the adapter you want to use for more specific configurations and instructions.

Adding new adapters is super easy and we are definitely looking for contributions on that front. Get in touch if you want to help!

Recipient

The Recipient Protocol enables you to easily make your structs compatible with Swoosh functions.

```
defmodule MyUser do
  @derive {Swoosh.Email.Recipient, name: :name, address: :email}
  defstruct [:name, :email, :other_props]
end
```

Now you can directly pass `%MyUser{}` to `from`, `to`, `cc`, `bcc`, etc. See `Swoosh.Email.Recipient` for more details.

Async Emails

Swoosh does not make any special arrangements for sending emails in a non-blocking manner. Opposite to some stacks, sending emails, talking to third party apps, etc in Elixir do not block or interfere with other requests, so you should resort to async emails only when necessary.

One simple way to deliver emails asynchronously is by leveraging Elixir's standard library. First add a Task supervisor to your application root, usually at `lib/my_app/application.ex`:

```
def start(_, _) do
  children = [
    ...,
    # Before the endpoint
    {Task.Supervisor, name: MyApp.AsyncEmailSupervisor},
    MyApp.Endpoint
  ]

  Supervisor.start_link(children, strategy: :one_for_one)
end
```

Now, whenever you want to send an email:

```
Task.Supervisor.start_child(MyApp.AsyncEmailSupervisor, fn ->
  %{name: "Tony Stark", email: "tony.stark@example.com"}
  |> Sample.UserEmail.welcome()
  |> Sample.Mailer.deliver()
end)
```

Please take a look at the official docs for Task and Task.Supervisor for further options.

One of the downsides of sending email asynchronously is that failures won't be reported to the user, who won't have an opportunity to try again immediately, and tasks by default do not retry on errors. Therefore, if the email must be delivered asynchronously, a safer solution would be to use a queue or job system. Elixir's ecosystem has many job queue libraries.

- Oban is the current community favourite. It uses PostgreSQL for storage and coordination.
- Exq uses Redis and is compatible with Resque / Sidekiq.

Phoenix integration

If you are looking to use Swoosh in your Phoenix project, make sure to check out the phoenix_swoosh project. It contains a set of functions that make it easy to render the text and HTML bodies using Phoenix views, templates and layouts.

Taking the example from above the "Getting Started" section, your code would look something like this:

> *web/templates/layout/email.html.eex*

```
<html>
  <head>
    <title><%= @email.subject %></title>
  </head>
  <body>
    <%= @inner_content %>
  </body>
</html>
```

> *web/templates/email/welcome.html.eex*

```
<div>
  <h1>Welcome to Sample, <%= @username %>!</h1>
</div>
```

> *web/emails/user_email.ex*

```
defmodule Sample.UserEmail do
  use Phoenix.Swoosh, view: Sample.EmailView, layout: {Sample.LayoutView, :email}

  def welcome(user) do
    new()
    |> to({user.name, user.email})
    |> from({"Dr B Banner", "hulk.smash@example.com"})
    |> subject("Hello, Avengers!")
    |> render_body("welcome.html", %{username: user.username})
  end
end
```

Feels familiar doesn't it? Head to the phoenix_swoosh repo for more details.

Attachments

You can attach files to your email using the `Swoosh.Email.attachment/2` function. Just give the path of your file as an argument and we will do the rest. It also works with a `%Plug.Upload{}` struct, or a `%Swoosh.Attachment{}` struct, which can be constructed using `Swoosh.Attachment.new` detailed here in the docs.

All built-in adapters have support for attachments.

```
new()
|> to("peter@example.com")
|> from({"Jarvis", "jarvis@example.com"})
|> subject("Invoice May")
|> text_body("Here is the invoice for your superhero services in May.")
|> attachment("/Users/jarvis/invoice-peter-may.pdf")
```

Testing

In your `config/test.exs` file set your mailer's adapter to `Swoosh.Adapters.Test` so that you can use the assertions provided by Swoosh in `Swoosh.TestAssertions` module.

```
defmodule Sample.UserTest do
  use ExUnit.Case, async: true

  import Swoosh.TestAssertions

  test "send email on user signup" do
    # Assuming `create_user` creates a new user then sends out a `Sample.UserEmail.welcome` email
    user = create_user(%{username: "ironman", email: "tony.stark@example.com"})
    assert_email_sent Sample.UserEmail.welcome(user)
  end
end
```

Mailbox preview in the browser

Swoosh ships with a Plug that allows you to preview the emails in the local (in-memory) mailbox. It's particularly convenient in development when you want to check what your email will look like while testing the various flows of your application.

For email to reach this mailbox you will need to set your `Mailer` adapter to `Swoosh.Adapters.Local`:

```
# in config/dev.exs
config :sample, MyApp.Mailer,
  adapter: Swoosh.Adapters.Local
```

In your Phoenix project you can `forward` directly to the plug without spinning up a separate webserver, like this:

```
# in web/router.ex
if Mix.env == :dev do
  scope "/dev" do
    pipe_through [:browser]

    forward "/mailbox", Plug.Swoosh.MailboxPreview
  end
end
```

You can also start a new server if your application does not depends on Phoenix:

```
# in config/dev.exs
# to run the preview server alongside your app
# which may not have a web interface already
config :swoosh, serve_mailbox: true
```

```
# in config/dev.exs
# to change the preview server port (4000 by default)
config :swoosh, serve_mailbox: true, preview_port: 4001
```

When using `serve_mailbox: true` make sure to have `plug_cowboy` as a dependency of your app.

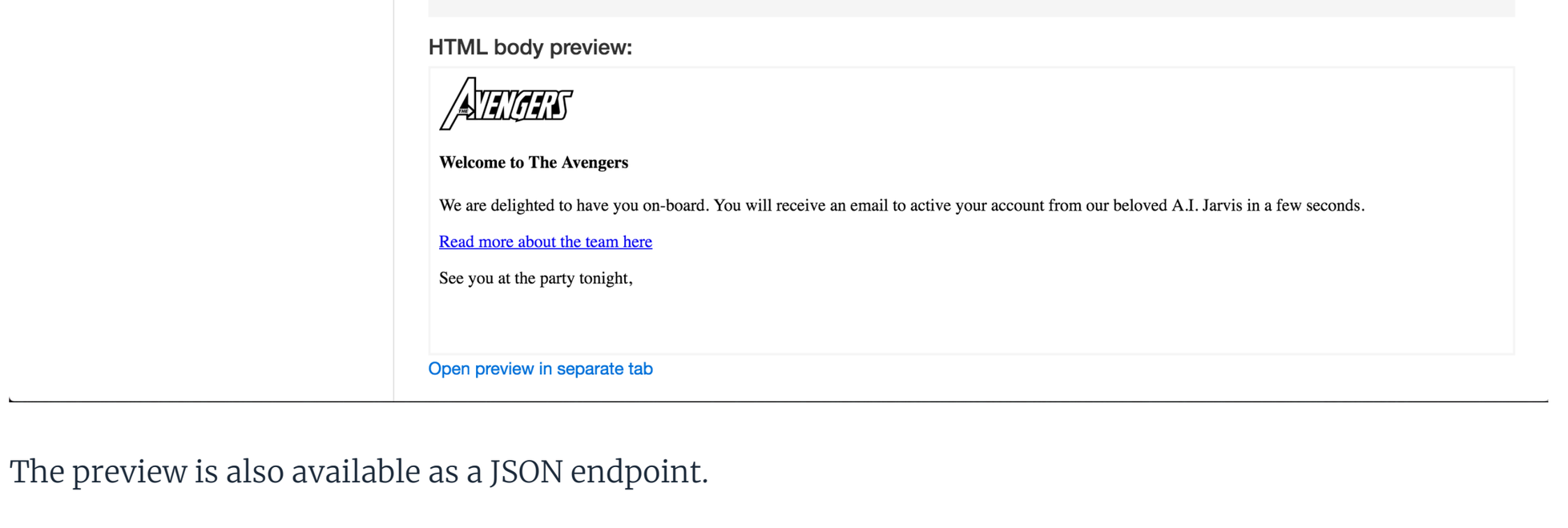
```
{:plug_cowboy, ">= 1.0.0"}
```

And finally you can also use the following Mix task to start the mailbox preview server independently:

```
$ mix swoosh.mailbox.server
```

Note: the mailbox preview won't display emails being sent from outside its own node. So if you are testing using an `IEx` session, it's recommended to boot the application in the same session. `iex -S mix phx.server` or `iex -S mix swoosh.mailbox.server` will do the trick.

If you are curious, this is how it the mailbox preview looks like:



The preview is also available as a JSON endpoint.

```
$ curl http://localhost:4000/dev/mailbox/json
```

Production

Swoosh starts a memory storage process for local adapter by default. Normally it does no harm being left around in production. However, if it is causing problems, or you don't like having it around, it can be disabled like so:

```
# config/prod.exs
config :swoosh, local: false
```

Telemetry

The following events are emitted:

- `[:swoosh, :deliver, :start]`: occurs when `Mailer.deliver/2` begins.
- `[:swoosh, :deliver, :stop]`: occurs when `Mailer.deliver/2` completes.
- `[:swoosh, :deliver, :exception]`: occurs when `Mailer.deliver/2` throws an exception.
- `[:swoosh, :deliver_many, :start]`: occurs when `Mailer.deliver_many/2` begins.
- `[:swoosh, :deliver_many, :stop]`: occurs when `Mailer.deliver_many/2` completes.
- `[:swoosh, :deliver_many, :exception]`: occurs when `Mailer.deliver_many/2` throws an exception.

View example in docs

Documentation

Documentation is written into the library, you will find it in the source code, accessible from `iex` and of course, it all gets published to HexDocs.

Contributing

We are grateful for any contributions. Before you submit an issue or a pull request, remember to:

- Look at our Contributing guidelines
- Not use the issue tracker for help or support requests (try StackOverflow, IRC or Slack instead)
- Do a quick search in the issue tracker to make sure the issues hasn't been reported yet.
- Look and follow the Code of Conduct. Be nice and have fun!

Running tests

Clone the repo and fetch its dependencies:

```
$ git clone https://github.com/swoosh/swoosh.git
$ cd swoosh
$ mix deps.get
$ mix test
```

Building docs

```
$ MIX_ENV=docs mix docs
```

LICENSE

See LICENSE