
River Admin Documentation

Release 0.6.0

Ahmet Dal

Mar 17, 2020

Contents

1	Donations	3
2	Live Demo	5
3	Getting Started	7
4	Contents	9
4.1	Getting Started	9
4.2	Authentication	12
4.3	Workflows	13
4.4	Functions	29
4.5	Custom Admin	30
4.6	Workflow Objects	32
4.7	Contribute	38
5	Indices and tables	41

River Admin is a very modern and a shiny customizable admin extension with user friendly and easy to use interfaces for [django-river](#). The power of it comes from the libraries it uses on both backend and frontend sides which are `django-river`, `django-rest-framework` `Vue` and `Vuetify`.

[River Admin Website](#)

CHAPTER 1

Donations

This is a fully open source project and it can be better with your donations.

If you are using River Admin to create a commercial product, please consider becoming our [sponsor](#) , [patron](#) or donate over [PayPal](#)

CHAPTER 2

Live Demo

<http://demo.riveradminproject.com/river-admin/>

- User: demo
- Password: demo

To run demo locally;

```
export LOCAL_DEMO=True
pip install -r requirements.txt
python manage.py migrate
python manage.py bootstrap_shipping_example
python manage.py bootstrap_issue_tracker_example
python manage.py bootstrap_river_admin_demo
python manage.py runserver
```

And then go to <http://127.0.0.1:8000/river-admin/>

Note: Create an admin user for yourself if you would like more access.

CHAPTER 3

Getting Started

You can easily get started with `django-river` by following *Getting Started*.

4.1 Getting Started

4.1.1 Requirements

- `django-river` $\geq$ 3.2.0
- Any Python version that is supported by `django-river`
- Any Django version that is supported by `django-river`
- Any browser that is supported by [Vuetify \(Browser Support\)](#)

4.1.2 Installation

Note: Before you can set up your workflow, your app integration with `django-river` must be done. Don't worry it is with the easiest setup. To see how to do it with `django-river` please have a look at *django-river*

1. Install and enable it

```
pip install river-admin
```

```
# settings.py

INSTALLED_APPS=[
    ...
    'river',
    'rest_framework.authtoken',
    'river_admin'
    ...
]
```

(continues on next page)

(continued from previous page)

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ],
    'EXCEPTION_HANDLER': 'river_admin.views.exception_handler'
}
```

2. Do migration;

```
python manage.py migrate
```

3. Register River Admin urls in your app urls.py

```
urlpatterns = [
    url(r'^$', include("river_admin.urls")),
]
```

4. Collect statics and make sure STATIC_URL is /static/ (**FOR PRODUCTION WHERE DEBUG=False**);

```
python manage.py collectstatic --no-input --no-post-process
```

5. Run your application;

```
python manage.py runserver 0.0.0.0:8000
```

6. Open it up on the browser and login with an admin user and enjoy the best way of flowing your work ever :-)

```
http://0.0.0.0:8000/river-admin/
```

4.1.3 Out of the Box Examples

River Admin comes with few examples that you can fiddle with and find your way easier.

Note: Enabling them will create their tables and also the necessary workflow components in the DB for you. It might be good idea to try them out on a development database.

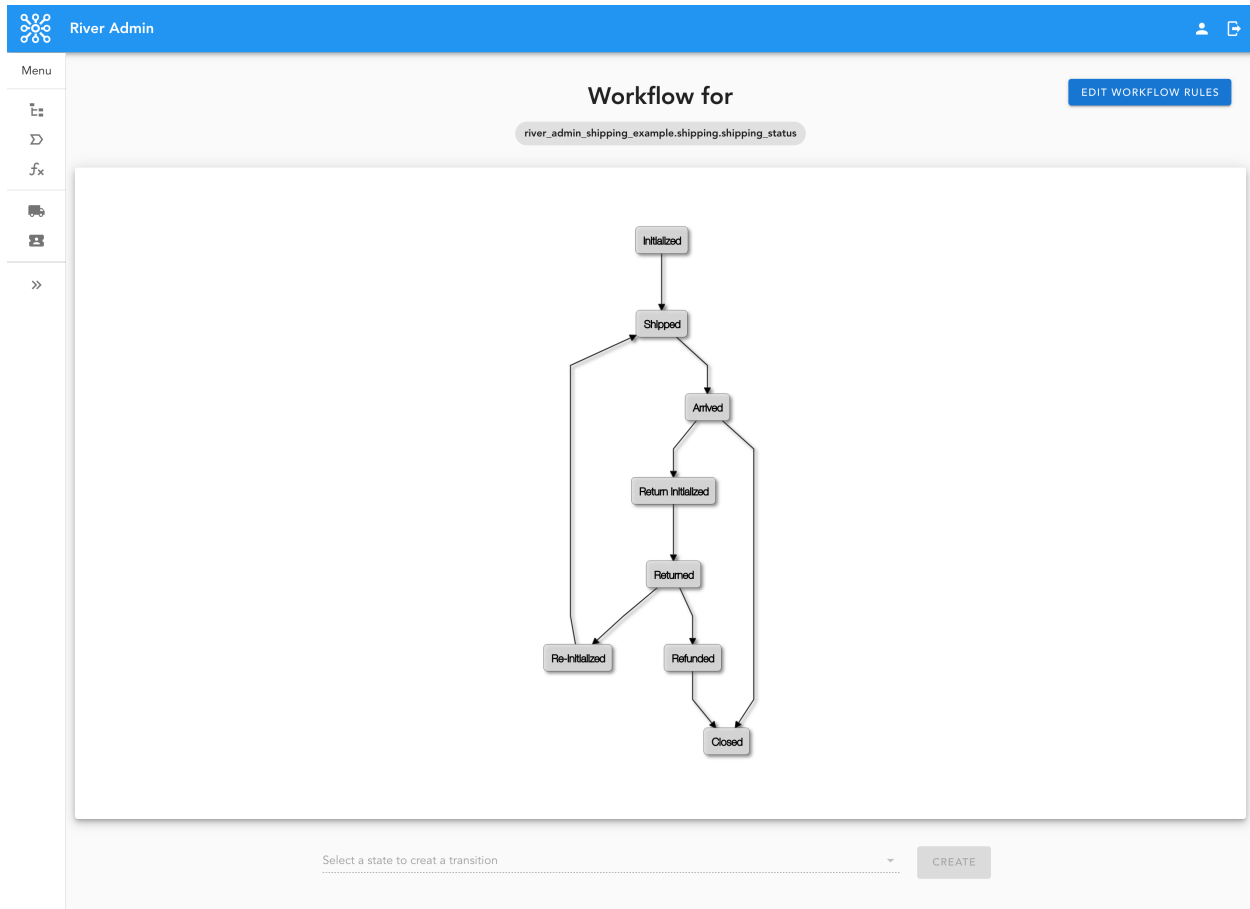
Shipping Flow

Enable the example app and then run your application

```
# settings.py

INSTALLED_APPS=[
    ...
    'river',
    'rest_framework.authtoken',
    'river_admin',
    'examples.shipping_example',
    ...
]
```

```
python manage.py migrate
python manage.py bootstrap_shipping_example
```



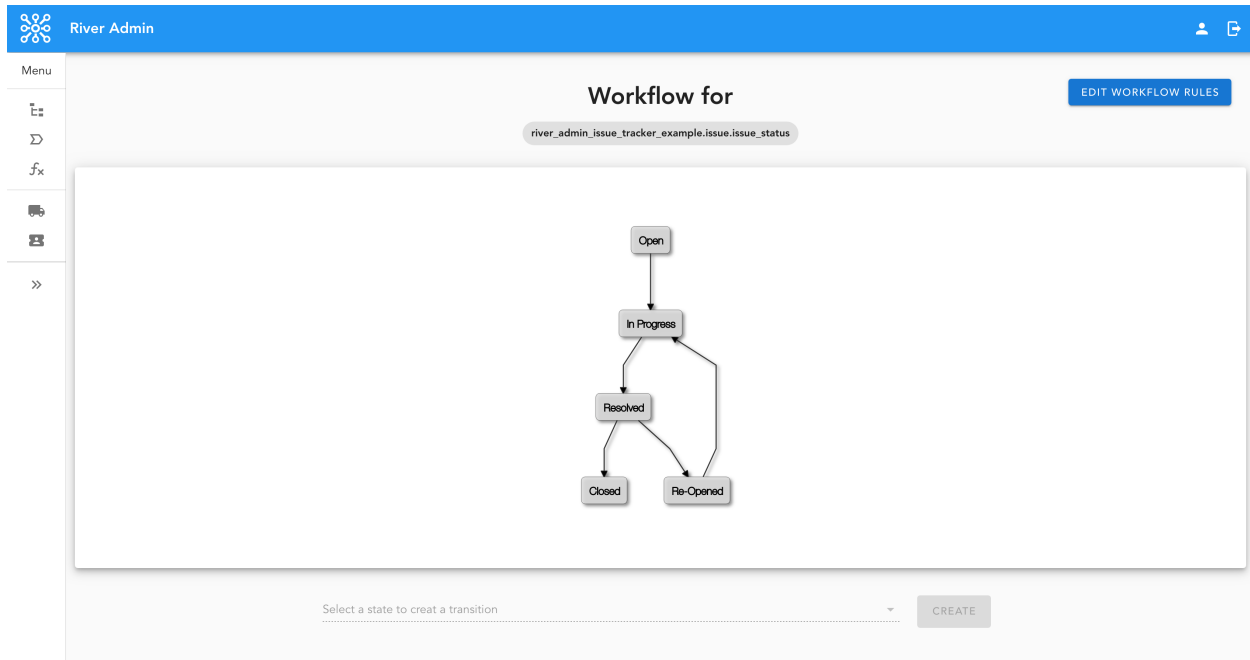
Issue Tracking Flow

Enable the example app and then run your application

```
# settings.py

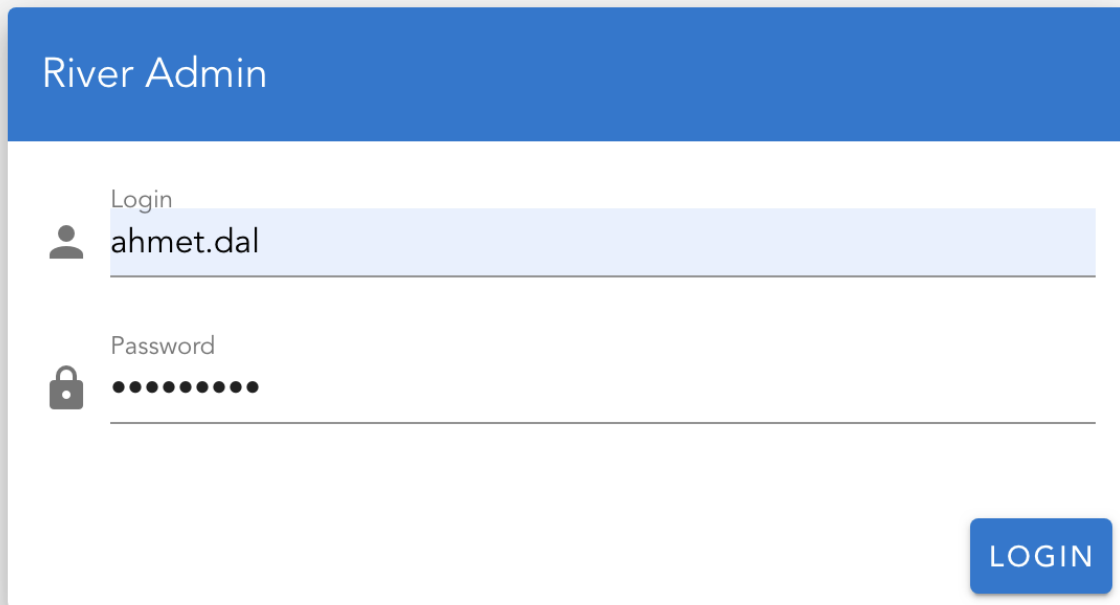
INSTALLED_APPS=[
    ...
    'river',
    'rest_framework.authtoken',
    'river_admin',
    'examples.issue_tracker_example',
    ...
]
```

```
python manage.py migrate
python manage.py bootstrap_issue_tracker_example
```



4.2 Authentication

River Admin is working like an administration interface and naturally it has an authentication mechanism. Even though it is working with the users who are created via Django API, it doesn't use Django Admin logged in your Django admin. So you will be asked to login with your user;

The image shows a login form for 'River Admin'. It has a blue header with the text 'River Admin'. Below the header, there are two input fields. The first is labeled 'Login' and contains the text 'ahmet.dal'. The second is labeled 'Password' and contains a series of dots. To the right of the password field is a blue button with the text 'LOGIN'.

Note: In order to be able to login, your user has to be either an admin user or having `river_workflow.view` permissions. So to speak, `river_workflow.view` is the minimum required permission to be able to use River Admin

Once user is logged in, they won't be asked by their credentials until they are logged out or their minimum required permission described above is revoked.

4.3 Workflows

4.3.1 Create Workflow

Note: Before you can set up your workflow, your app integration with `django-river` must be done. Don't worry it is with the easiest setup. To see how to do it with `django-river` please have a look at `django-river`

Note: In order to see this page, your user has to have `river.add_workflow` permission.

Creating a workflow consist of defining its initial state and the state field along with the content type and later on the transitions by just clicking.

Initiale Workflow

- if you haven't had any workflow before, Home page will wellcome you with a button which will take yo to the workflow creation page. If you have had a workflow before then just click the plus button on the right bottom corner to navigate to the page.



You don't seem to have any workflow

Why don't you create your first workflow?

BEGIN

- Pick the workflow that you have had the integration with `django-river`



Create your initial state

Creating your initial state, you'll be able to start designing your workflow.

Search for a workflow...

river

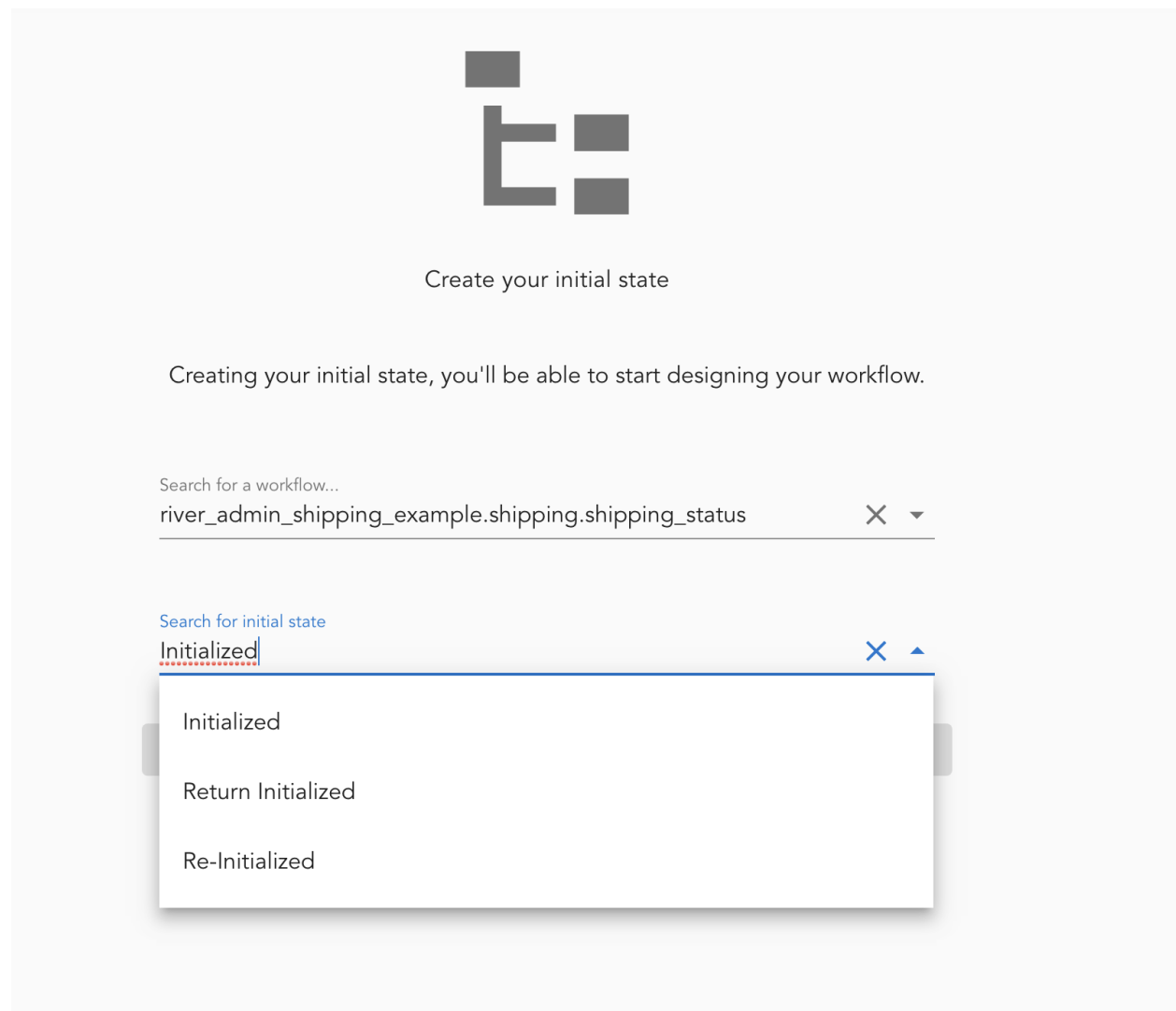


river_admin_shipping_example.shipping.shipping_status

Search for initial state

CREATE

- Pick the initial state of your workflow and lastly hit the create button



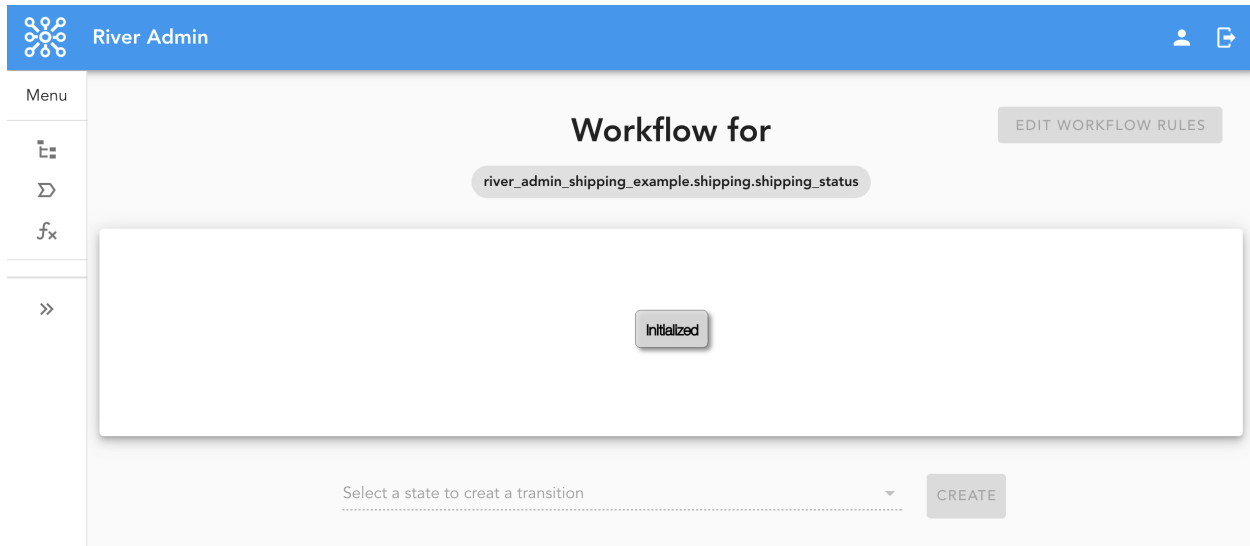
Create Transitions

This is the most fun part of River Admin. Because it provides a highly interactive one screen to create them all.

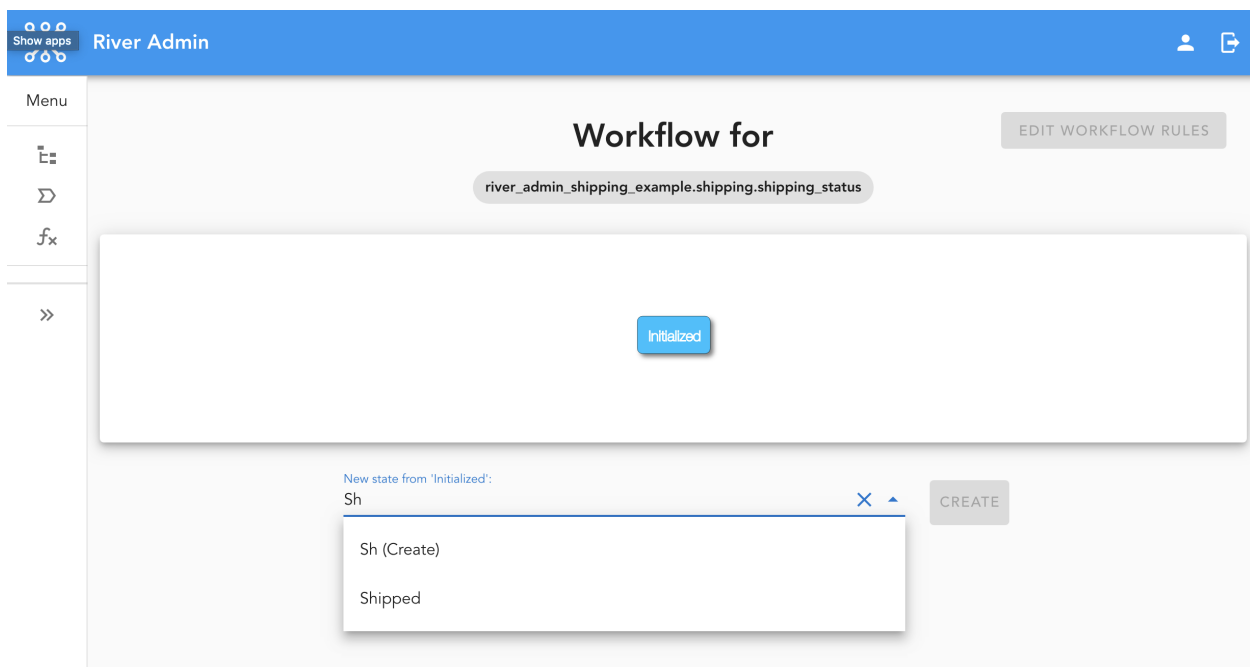
Note: Every change you make on that screen is an online action. Meaning that it will also be applied on the DB.

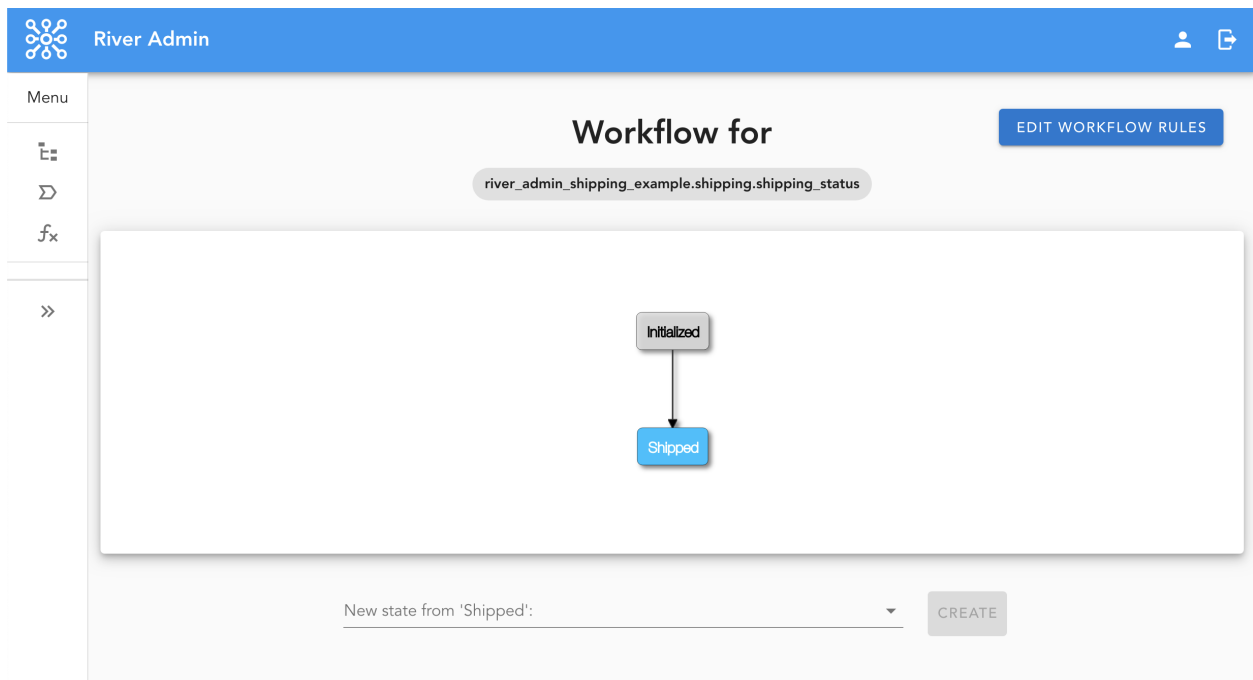
Note: Same screen can anytime be used to update the transitions in the workflow as well

- What you are going to run into after you have done the previous steps is something like this;



- This interface expects you to click the rectangles which are representing the transition states to create next state out of them. Simply click them and that will change their color so that you can understand you managed to select

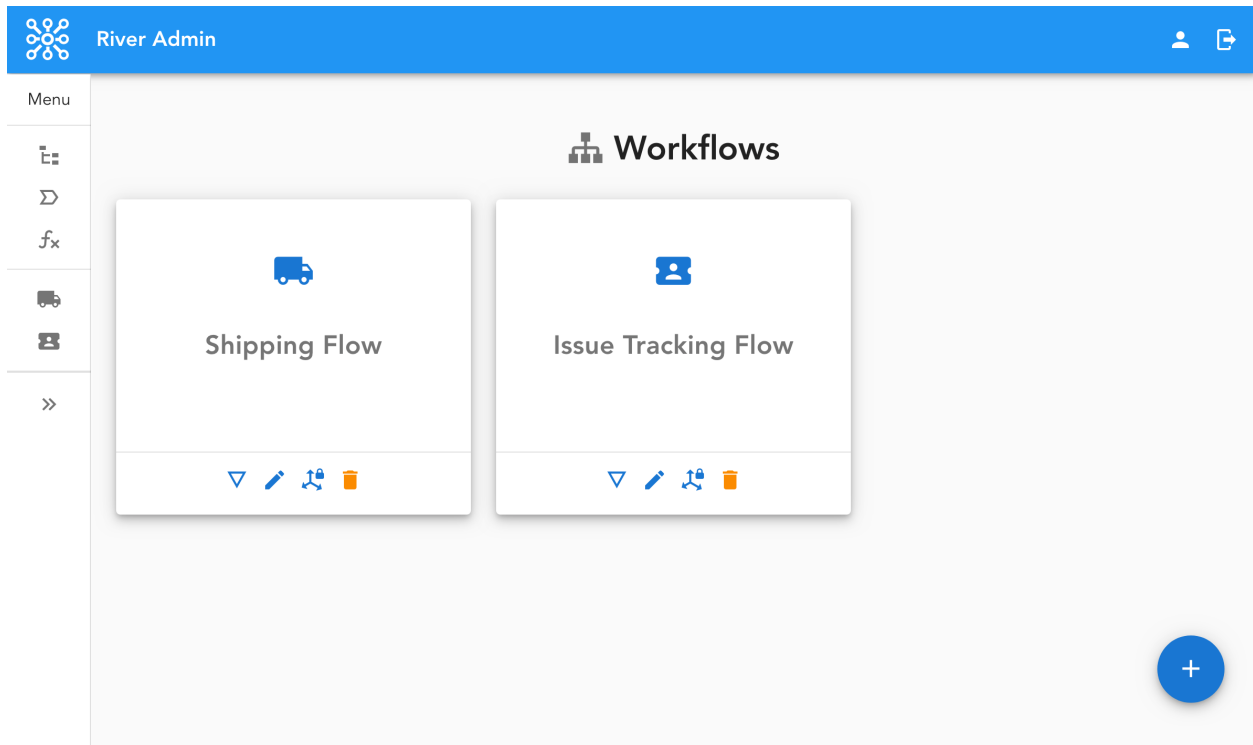




4.3.2 List Workflows

Note: In order to see this page, your user has to have `river.view_workflow` or `river.change_workflow` in older version of Django.

Home page of River Admin is also listing workflow page. If you are home for the first time meaning that you haven't had any workflow before, this page won't have any workflow to list. But once you have your first workflow you will start seeing them.



If your user is authorized to do those operations, the component comes with bunch of button that allows you to go to a workflow's detail, editing the workflow, authorization rule management pages and a button to delete it.



Note: Those buttons for the workflow will be disabled if you don't have the required permissions.

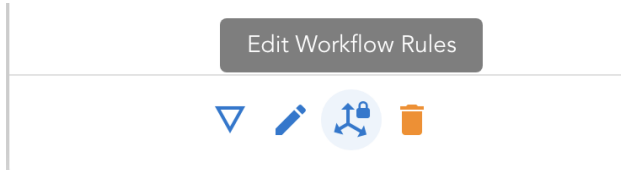
4.3.3 Authorizations

Thanks to `django-river` because it supports flexibility to manage workflow components on the fly meaning that the changes can be applied without a code change or a deployment. `River Admin` is extending this capability and providing a user friendly, easy to use screens to do that on top of `django-river`. One of these screens is managing the authorization rules of the transitions.

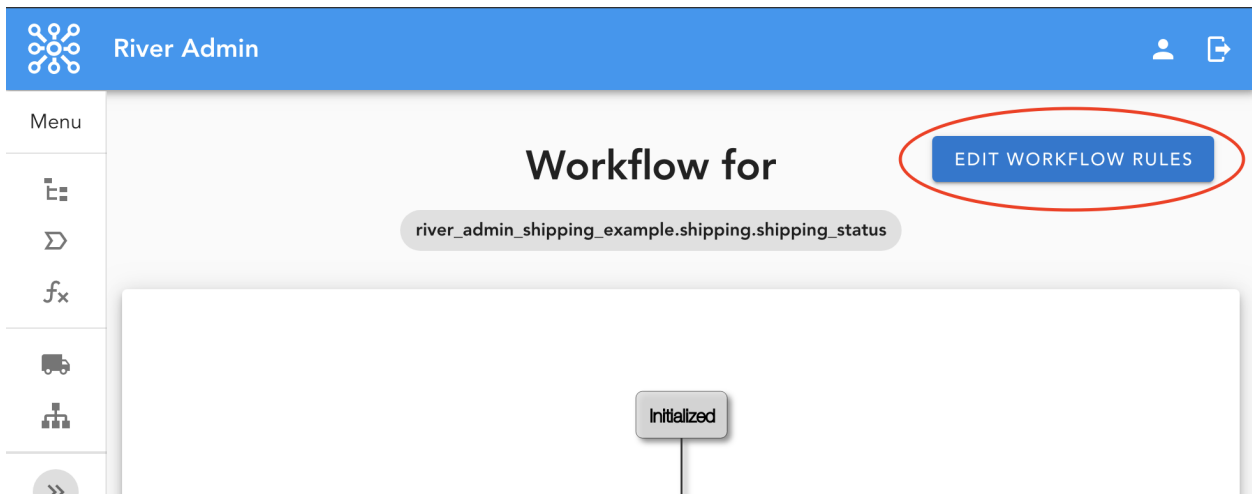
How to Navigate

Note: In order to see this page, your user has to have `river.view_workflow` together with `river.change_workflow` permissions.

In order to go to that screen you can either be navigated from the home screen by clicking the “Edit Workflow Rules” button;

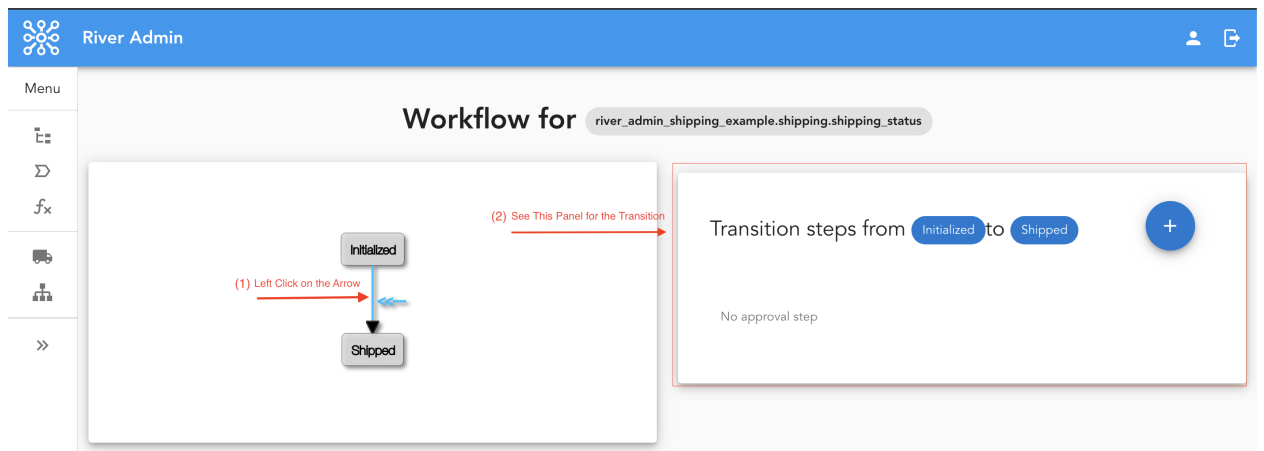


Or from the editing workflow page by clicking “Edit Workflow Rules” button;



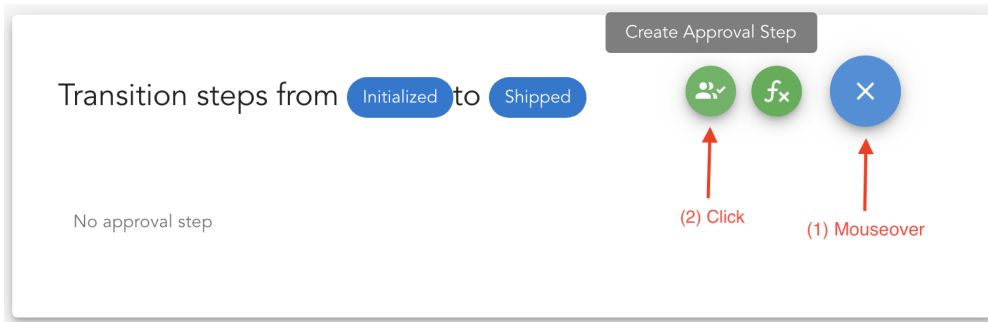
Add Authorization Rule

River Admin provides a graphical interface that illustrates states and transitions in the workflow which you have already seen while creating the workflow. Same component exists here but with more capability. That extra capability you have here is to be able to click transitions which are illustrated with arrows between the states. Later you will see another component on the right side of the screen where you can manage the authorization rules for the transition.



Step 1:

Step 2:



Step 3:

New Approval Rule

Search permissions

Search groups

Deli

Delivery Person

CREATE APPROVAL

Step 4:

New Approval Rule

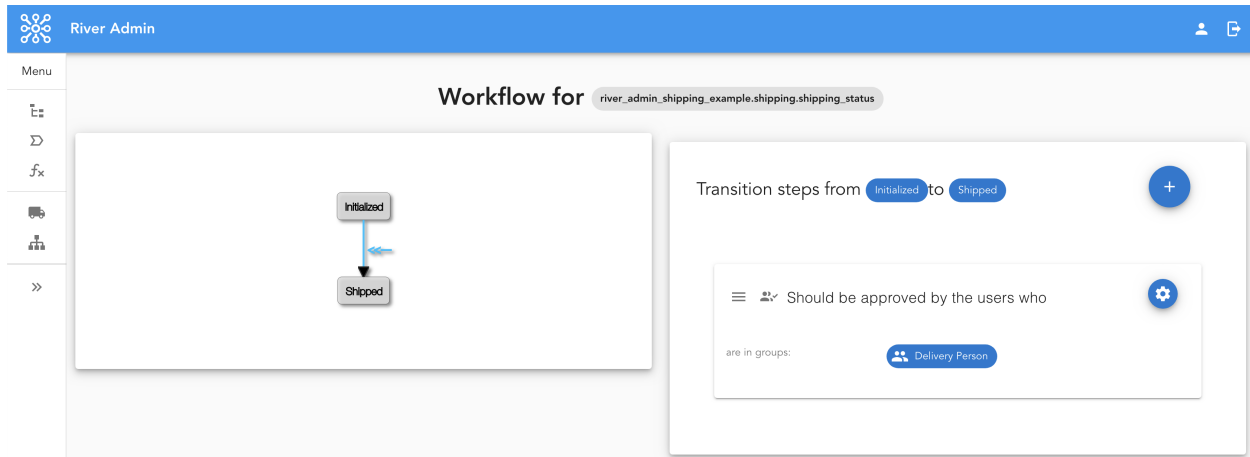
Search permissions

Search groups

Delivery Person

CREATE APPROVAL

After the authorization rule is created successfully;

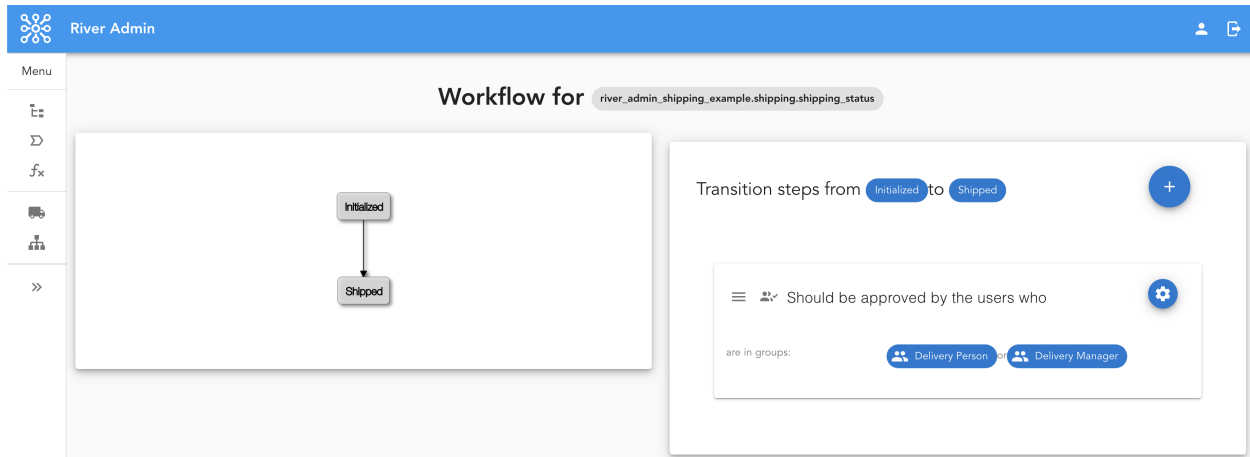


The authorization rule we have just created means that in order the transition to happen a user within *Delivery Person* user group should approve it.

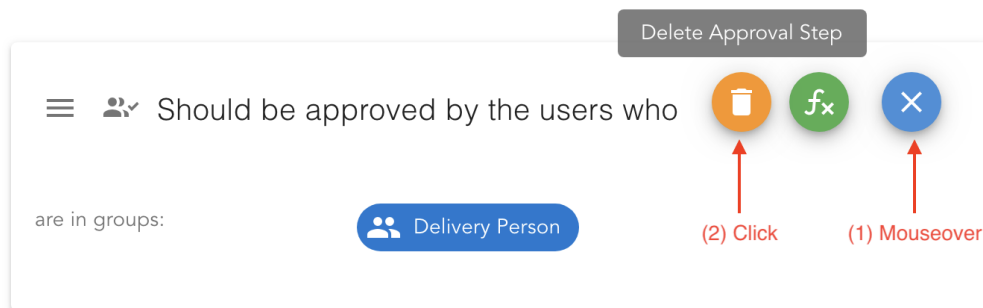
Multiple Groups

Multiple user groups can also be selected in one authorization rules as it is already supported by *django-river* and that would mean that anyone who is in those groups can approve the transition;

The screenshot shows the 'New Approval Rule' form in the River Admin interface. The form has a title 'New Approval Rule'. Below the title is a 'Search permissions' field with a dropdown arrow. Below that is a 'Search groups' field with a dropdown arrow. The 'Search groups' field shows two selected groups: 'Delivery Person' and 'Delivery Manager'. Below the 'Search groups' field is a 'CREATE APPROVAL' button.



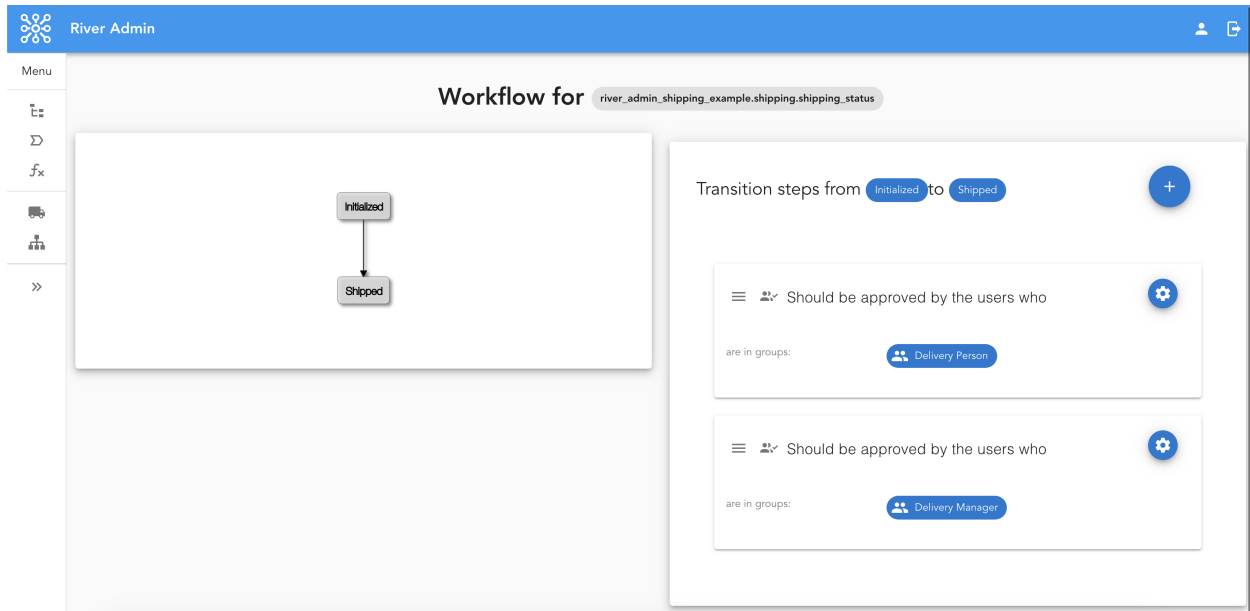
Delete Authorization Rule



Chain of Authorization Rules

This is one of the cool feature of `django-river` already. Multiple authorization rules can be chained together with a prioritization mechanism. With this a first authorization rule should be satisfied before the second one can kick in.

Note: This should not be mixed up with having multiple groups in one authorization rule. Because with multiple groups anytime any user in any of the specified group is authorized to approve the transition.



What is created in the image above is a chain of authorization rules for the transition. It means that a users within `Delivery Person` group should first approve it before it is on the user's approval who are in the `Delivery Manager` group.

Note: The prioritization order matters here.

Reprioritization

One of the most convenient improvement with `River Admin` on top of `django-river` is changing the order of the chain by just a drag and drop.



4.3.4 Hooks

Hooks are the workflow component for you to subscribe certain events in your workflow like a transition happens, an authorization rule is approved or the workflow is completed. It can be either a rule in general for whole workflow or for a specific workflow object too. In this section, we will be looking into the first one

After `django-river` version `3.0.0`, hooks are supported on the fly and this is another feature of `django-river` which `River Admin` has some interfaces for.

Hooks are managed on the same page with authorizations for a workflow. So, to know how to navigate to the page, please take a look at [How to Navigate](#)

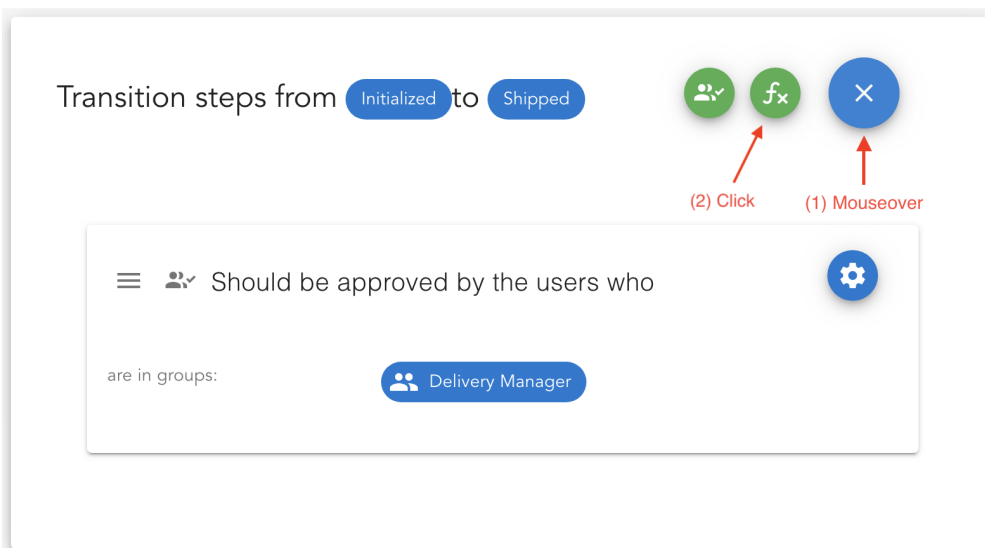
Note: To register a hook to a certain event, you need the required function to be defined preliminary. To see how to create a function please look at [Create & Update Function](#)

Note: Even though `django-river` supports transition, approvals and workflow completing events to hook up, `River Admin` currently supports only transitions and approval hooks.

Apart from that even though `django-river` gives you the capability of defining your hooks before the event happens or after the event happens, `River Admin` simplifies this by just allowing before transition happens and after an authorization rule is approved for the sake of convenience. By this it is very smooth to create your workflow along with your hooks all together.

Create Transition Hooks

Transition hooks can be defined for whole transition. It means that the hook will be executed right before the transition happens.



New Hook

Search functions

Req

Request new stock if it is running out

CREATE HOOK

New Hook

Search functions

fx Request new stock if it is running out

CREATE HOOK

After you create, it should look like this;

Transition steps from **Initialized** to **Shipped**

Should be approved by the users who

are in groups: **Delivery Manager**

Right before the transition happens

fx Execute function **Request new stock if it is running out**

Create Approval Hooks

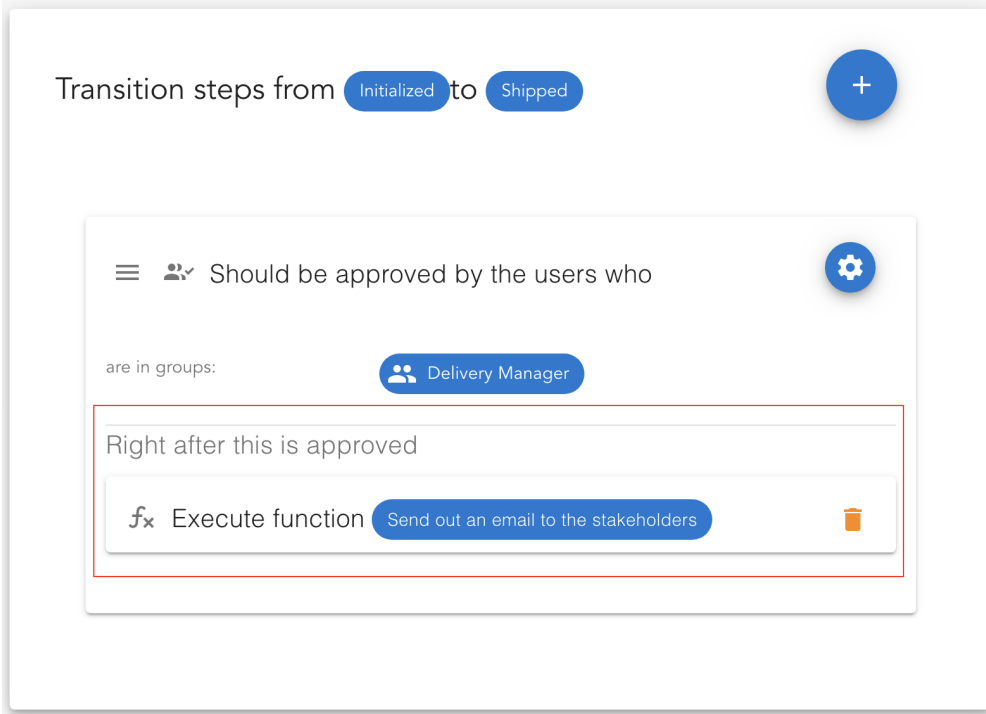
Approval hooks can be defined for a specific authorization rule not for whole transition. After the authorization rule is approved your hook will be invoked.

The first screenshot shows a transition from 'Initialized' to 'Shipped' with a '+', and a rule 'Should be approved by the users who are in groups: Delivery Manager'. It highlights three icons: a trash can, a green 'fx' icon (labeled '(2) Click'), and a blue 'X' icon (labeled '(1) Mouseover').

The second screenshot, titled 'New Hook', shows a search for functions with the text 'Se' and a dropdown list containing 'Send out an email to the stakeholders'. A 'CREATE HOOK' button is at the bottom right.

The third screenshot, also titled 'New Hook', shows the same search results with the 'Send out an email to the stakeholders' option selected and highlighted. The 'CREATE HOOK' button remains at the bottom right.

After you create, it should look like this;



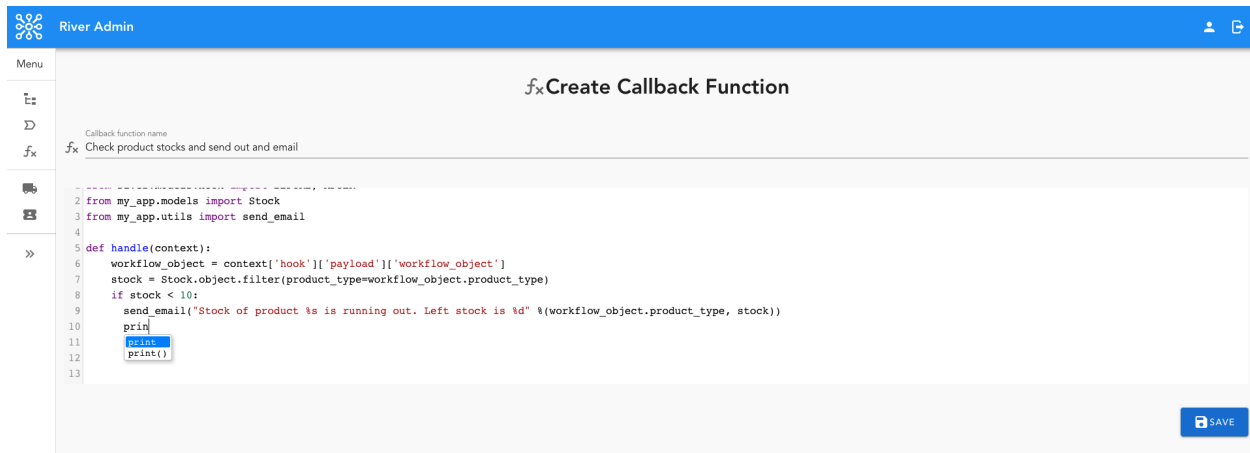
4.4 Functions

`django-river` after version 3.0.0 started to support on the fly functions and the hooks on certain events in a workflow. River Admin is making it very easy to use with some fancy user interfaces. Here how you can manage your functions;

4.4.1 Create & Update Function

Note: River Admin is not extending the APIs of `django-river`. In order to see how your functions should look like please visit the [django river function documentation](#) itself

Note: In order to see this page, your user has to have `river.add_function` permission.

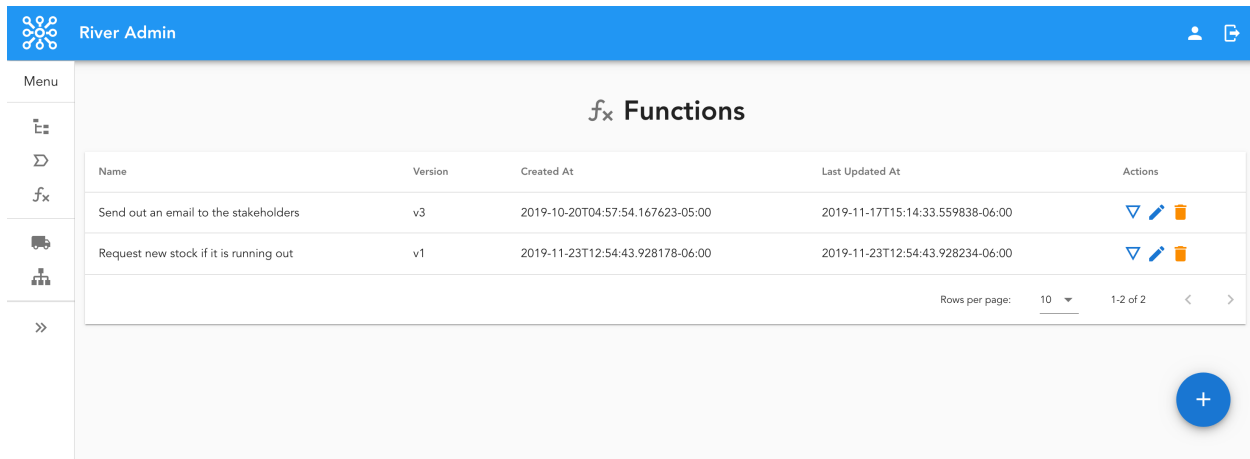


Note: Changes on the functions are applied to your hooks right away. It means that, next time a hook is kicked in with this function will be executed with the up to date version of the function.

4.4.2 List Functions

Note: In order to see this page, your user has to have `river.view_function` or `river.change_function` permissions in older version of Django.

On this page you can be navigated to the create page by clicking the button with the plus icon on the right bottom corner of the screen and viewing and editing the function pages by clicking the action button of the corresponding function. You can also delete a function on this page.



4.5 Custom Admin

River Admin is also meant to be functioning like any other django admin libraries but specifically for workflow operations. You can definitely be using River Admin as it comes but there are quite cool customizations you can do with River Admin. The way how to customize River Admin is very much like the way how you customize your Django model admin. We kept the same practice

```
# admin.py

import river_admin
from examples.shipping_example.models import Shipping

class ShippingRiverAdmin(river_admin.RiverAdmin):
    name = "Shipping Flow"
    icon = "mdi-truck"
    list_displays = ['pk', 'product', 'customer', 'shipping_status']

river_admin.site.register(Shipping, "shipping_status", ShippingRiverAdmin)
```

Note: River Admin users material icon sets. In order to see what icons you can use more please take a look at [Material Design Icons](#). What ever you want to use from there just add mdi- prefix to the icon name.

Here is the output;

The screenshot displays the River Admin interface. On the left is a sidebar menu with options: Workflows, States, Functions, and Shipping Flow (selected). The main area shows a 'Workflows' section with a card for 'Shipping Flow' featuring a truck icon and a list of actions (view, edit, refresh, delete). Below this is a table titled 'Workflow objects of river_admin_shipping_example.shipping shipping_status'.

pk	product	customer	shipping_status	Actions
4	iPhone 11 Pro	Ahmet Dal	Initialized	
5	MacBook Pro 15"	Ahmet Dal	Shipped	

At the bottom right of the table, it shows 'Rows per page: 10' and '1-2 of 2'.

4.5.1 Method Field

River Admin supports custom field that can be fetched from a python method instead of the workflow object itself like in Django model admins.

```
# admin.py

import river_admin
from examples.shipping_example.models import Shipping

class ShippingRiverAdmin(river_admin.RiverAdmin):
    name = "Shipping Flow"
    icon = "mdi-truck"
    list_displays = ['custom_pk', 'product', 'customer', 'shipping_status']

    @classmethod
    def custom_pk(cls, obj):
        return "Primary Key: %d" % obj.pk

river_admin.site.register(Shipping, "shipping_status", ShippingRiverAdmin)
```

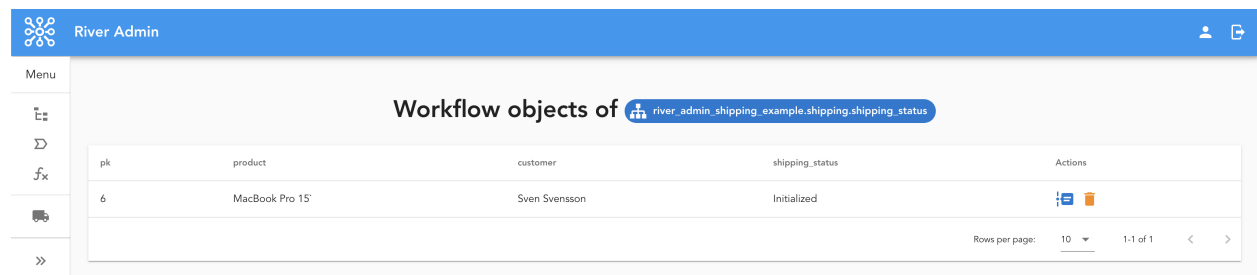
4.6 Workflow Objects

Workflow objects are active objects that are tied to the workflow specification you created. There are some more user interfaces for those workflow objects like the timeline of one and also customize the lifecycle of one with new hooks.

4.6.1 List Workflow Objects

Note: In order to see this page, your user has to have `river.view_workflow` or `river.change_workflow` in older version of Django.

After you are navigated by clicking your workflow icon on the left panel you will be run into a listing page where all of your workflow objects are listed. Then you can go to each of theirs timeline pages or you can delete the workflow objects.



The screenshot shows the River Admin web interface. At the top is a blue header with the River Admin logo and name, and user/logout icons. On the left is a sidebar menu. The main content area is titled "Workflow objects of" followed by a blue pill containing a truck icon and the text "river_admin_shipping_example.shipping.shipping_status". Below this is a table with columns: pk, product, customer, shipping_status, and Actions. The table contains one row with data: pk=6, product=MacBook Pro 15', customer=Sven Svensson, shipping_status=Initialized. The Actions column has icons for edit and delete. At the bottom right of the table, it says "Rows per page: 10" and "1-1 of 1" with navigation arrows.

pk	product	customer	shipping_status	Actions
6	MacBook Pro 15'	Sven Svensson	Initialized	 

Rows per page: 10 1-1 of 1 < >

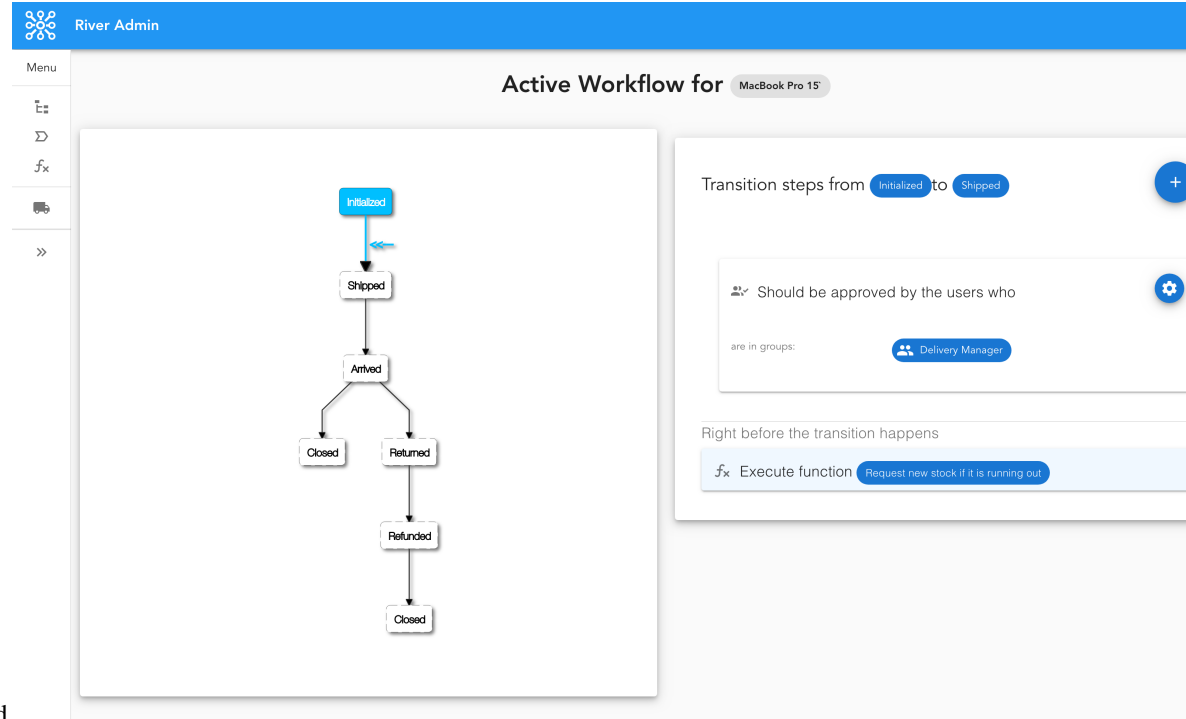
Note: To customize which columns of the workflow objects should show up on this interface, please look at [Custom Admin](#).

4.6.2 Timeline

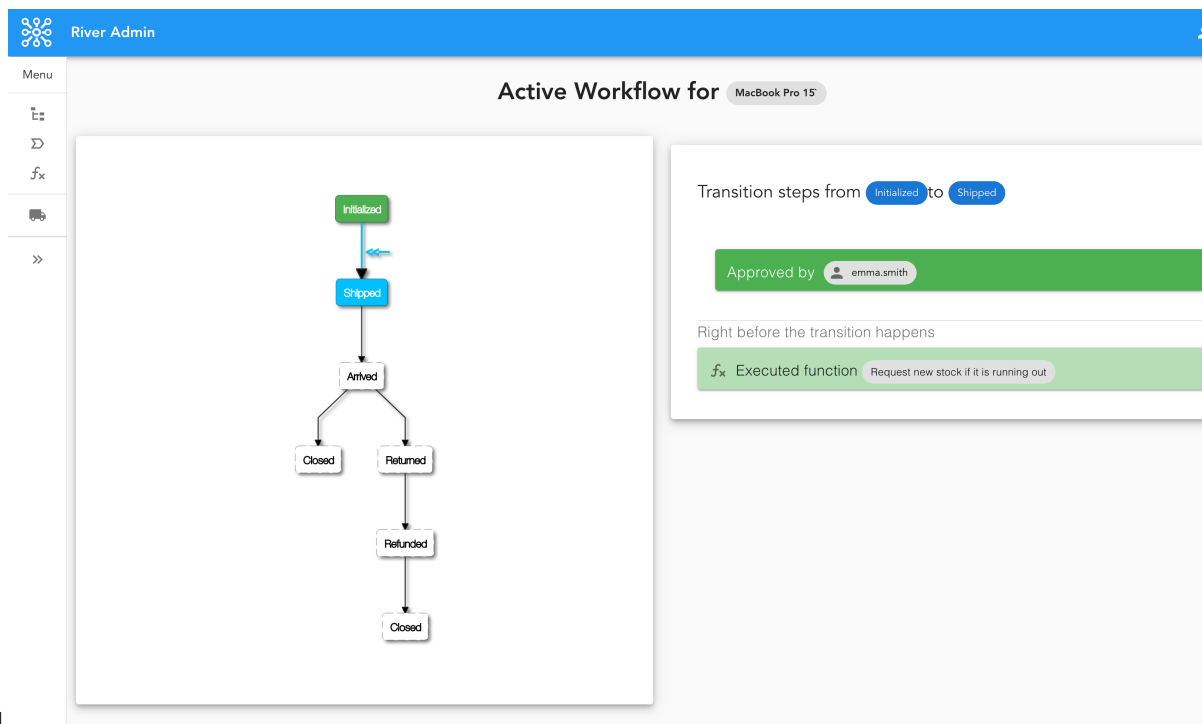
Each workflow object in `django-river` has a different instance of the the same lifecycle. The transitions, authorization rules and hooks are created out of the workflow specification once and special to the workflow object itself.

River Admin is showing the timeline of one workflow object that is consist of states, transitions, authorizations and hooks that scattered around in one useful user interface. On this page you can see what has happened so far along with what will possibly happen with the object. You can even add a transition and approval hooks tied to a particular object that is not for whole workflow object.

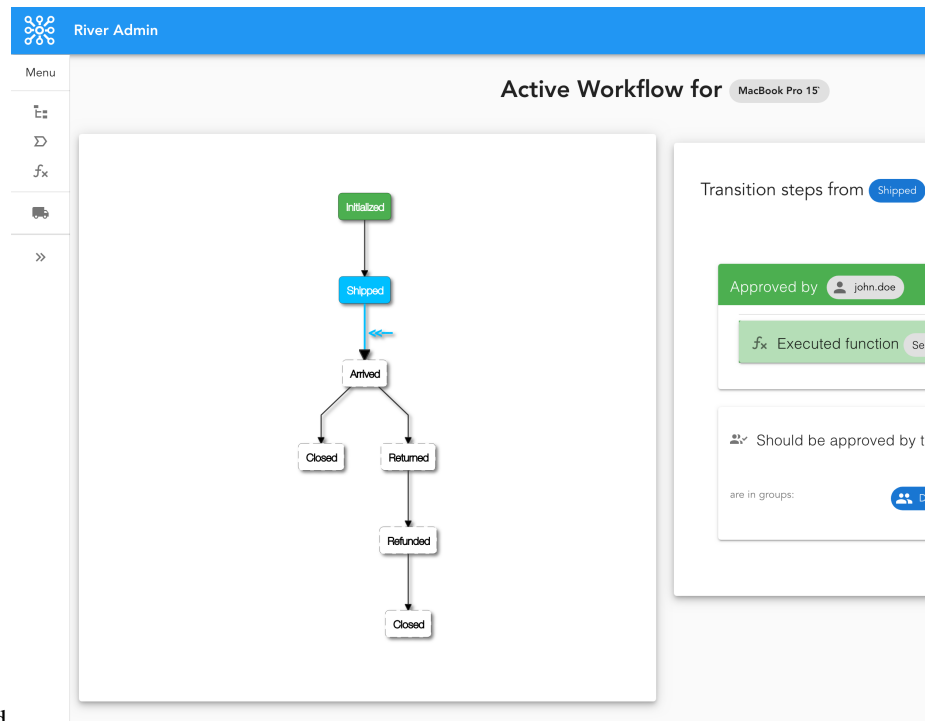
Here you can see an example of whole lifecycle of a workflow object;



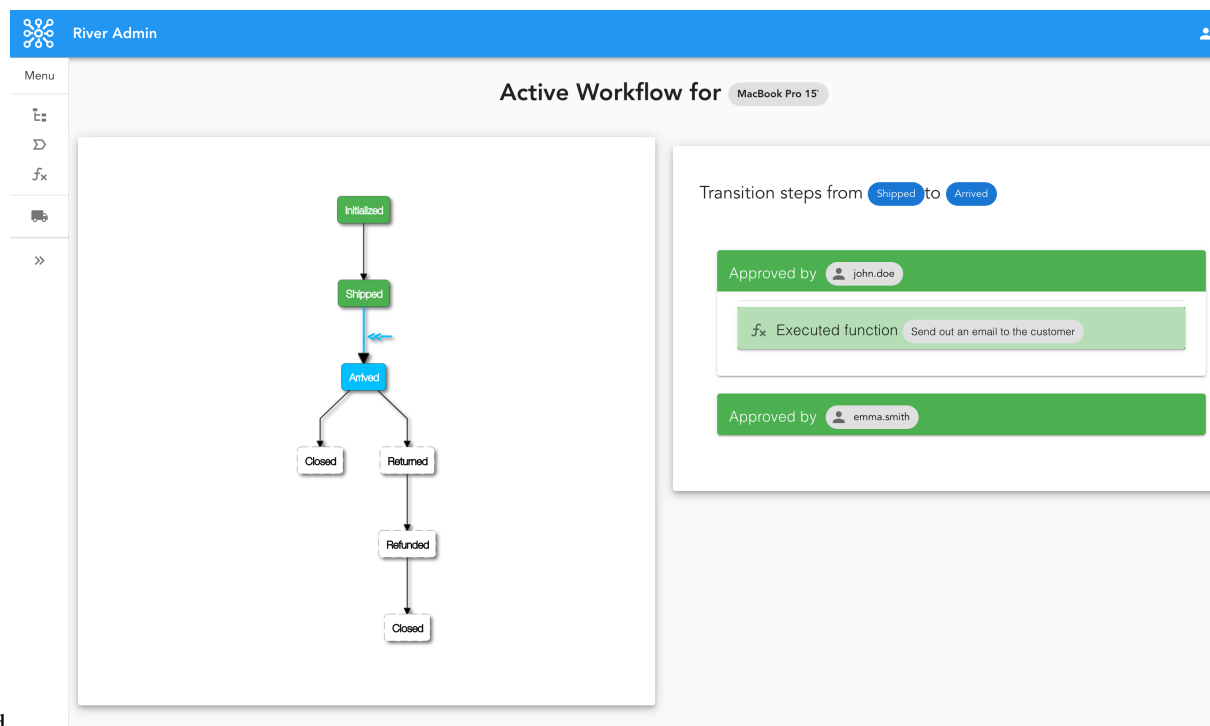
Current State: Initialized



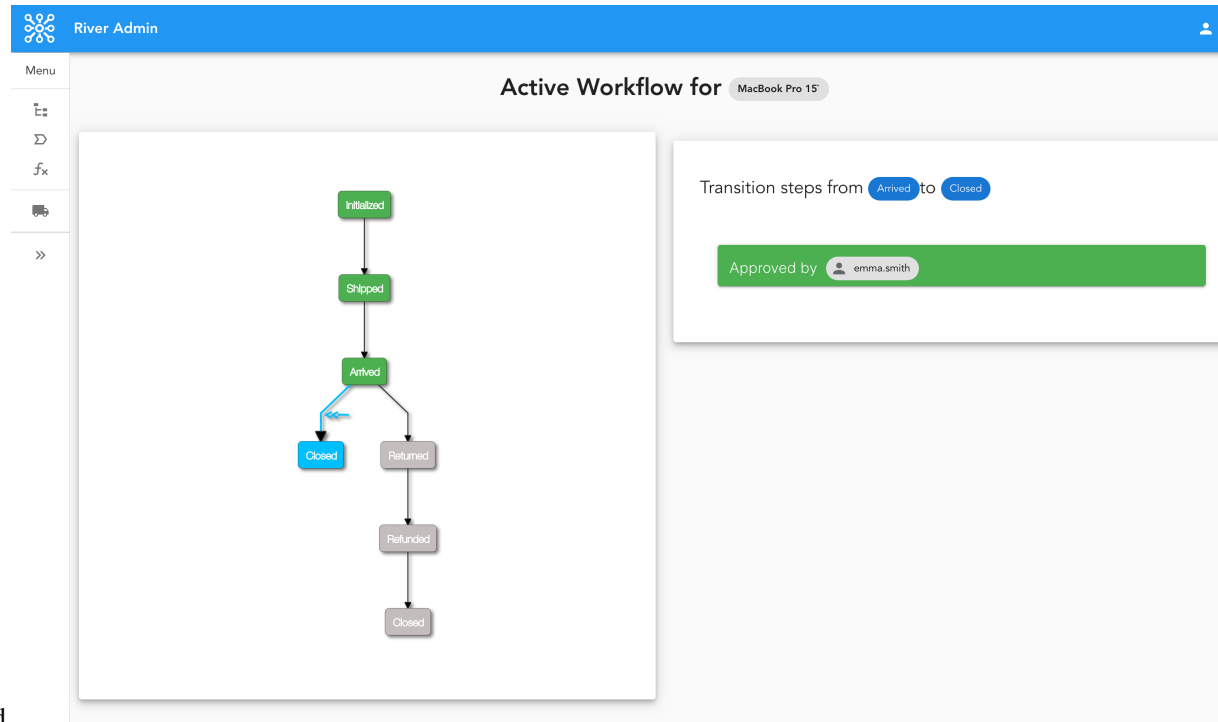
Current State: Shipped



Current State: Arrived Approved but still Shipped



Current State: Arrived



Current State: Closed

Colors

There are different type of indicator colors for different states like current state (Blue), approved state (Green), possible state (White with dashed border) or impossible state (Grey).

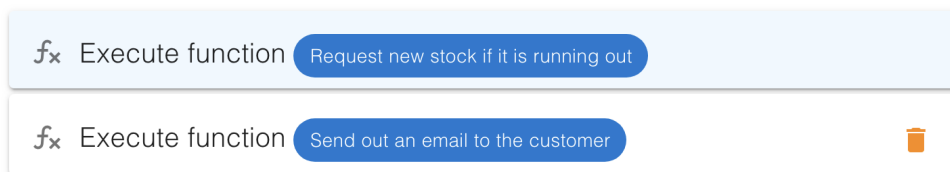


4.6.3 Object Hooks

Note: What hooking is described in workflow level [Hooks](#). So please visit there to get the idea of the hooks.

Object level hooking is tied to a specific object unlike workflow level hookings. It means the the hooks that you created for an object won't be kicked in for the other objects that are same kind.

On the timeline page, those you created at the workflow level will be shown with `aliceblue` color and without a delete button since it is a general hook for all objects in the workflow. But you will see object level hooks with white color and a delete button as shown below;



Create Transition Hooks

The top screenshot shows a 'Create Transition Hook' dialog. It displays a transition from 'Initialized' to 'Shipped'. A hook configuration box contains the text 'Should be approved by the users who are in groups: Delivery Manager'. Below this, it says 'Right before the transition happens' and shows a hook configuration: 'Execute function Request new stock if it is running out'. Red arrows point to the 'fx' and 'x' icons with labels '(2) Click' and '(1) Mouseover' respectively.

The middle screenshot, titled 'New Hook', shows a search for functions with the text 'custo' in the search bar. A dropdown menu shows 'Send out an email to the customer'. A 'CREATE HOOK' button is visible.

The bottom screenshot, also titled 'New Hook', shows the same search results. The function 'Send out an email to the customer' is now selected and highlighted in a blue box. The 'CREATE HOOK' button is still present.

After you create, it should look like this;

Transition steps from **Initialized** to **Shipped** 

 Should be approved by the users who

are in groups: 

Right before the transition happens

 Execute function **Request new stock if it is running out**

 Execute function **Send out an email to the customer** 

Create Approval Hooks

 Should be approved by the users who

are in groups: 

Create Approval Hook

(2) Click (1) Mouseover

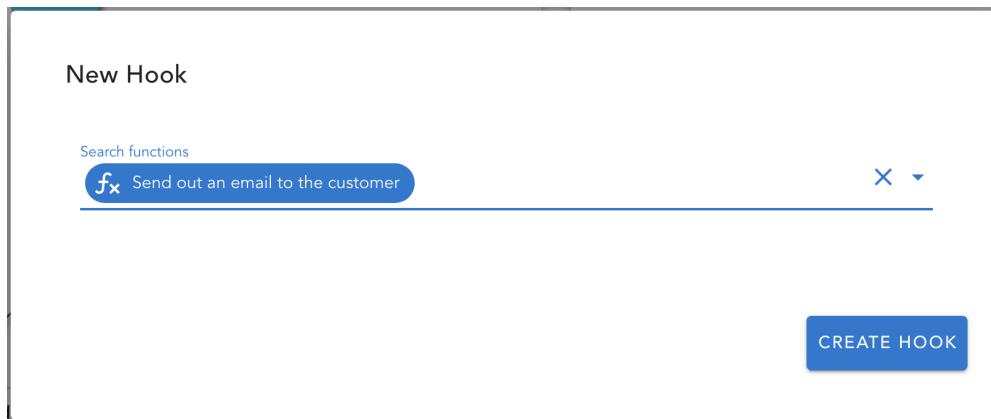
New Hook

Search functions

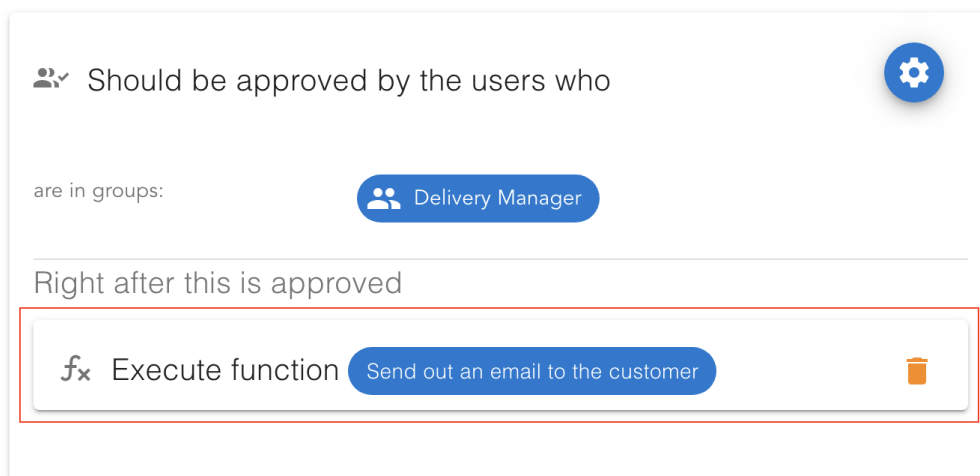
custo

Send out an email to the customer

CREATE HOOK



After you create, it should look like this;



4.7 Contribute

River Admin consists of two parts that are backend and ui. It is built with [Django Rest Framework](#) on the backend side whereas [Vue](#) on the front end side.

4.7.1 Rive Admin Backend

River Admin backend side is built with Django and [Django Rest Framework](#). So you need Django development environment to be set up.

1. Install the dependencies

```
pip install -r requirements.txt
```

2. Install [Tox](#) to run the tests for all environments.

```
pip install tox
```

3. Install [Twine](#) and other necessary packages to upload packages to PyPI

```
pip install wheel setuptools twine
```

4. You are ready to develop the backend side now.

Development

```
python manage.py runserver
```

Tests

`Tox` is used on the backend side to automate Python & Django testing. Tests are under `river-admin/tets/`. Simply run

```
tox
```

To run it for a specific environment;

```
tox -e py34-dj2.1
```

4.7.2 River Admin UI

River Admin ui is built with `Vue`. So you need Vue development environment to be set up.

1. Install node & npm
2. Install yarn ([Install Yarn](#))
3. Install dependencies

```
yarn install
```

Development

While developing the front end Vue app of River Admin, you can run it without building it and the server will automatically reload on any changes in the code. This is quite useful thing for fast feedback and debugging. One thing you should make sure before you run this, backend server is also running since it needs to call the backend

```
python manage.py runserver
```

```
yarn serve
```

Tests

UI tests are written with `Jest` javascript testing framework from Facebook. Tests are under `ui/tets/`. To run the tests simply;

```
yarn test:unit
```

To run a specific one;

```
yarn test:unit StateInput.spec.js
```

To run the tests with a fresh snapshot (to clean the snapshots);

```
yarn test:unit -u
```

Build

```
yarn build
```

The distribution folders of the Vue app are `river_admin/templates` and `river_admin/static`. The reason for that is because a Django app should contains all the html and static files under `templates` and `static` folders.

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`