Creating a modern web application using Symfony API Platform, ReactJS and Redux

by Jesus Manuel Olivas & Eduardo Garcia
@jmolivas | @enzolutions

Who We Are?

Jesus Manuel Olivas

jmolivas@weknowinc.com

jmolivas

jmolivas

http://drupal.org/u/jmolivas

http://jmolivas.weknowinc.com

weKnow

Who We Are?

Eduardo Garcia

enzo@weknowinc.com

enzolutions

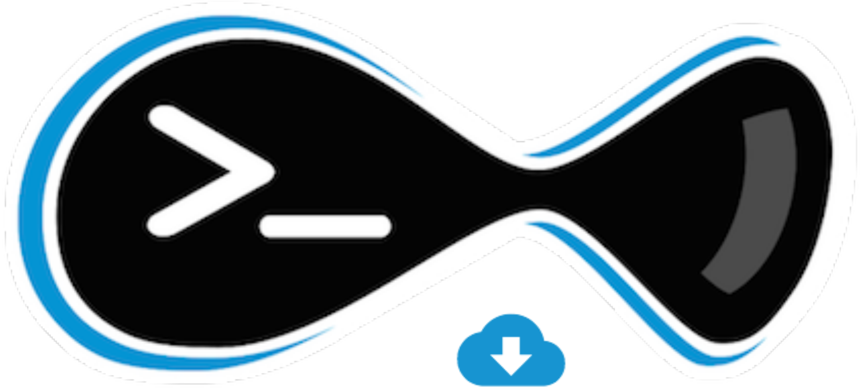
enzolutions

http://drupal.org/u/enzo

http://enzolutions.com

weKnow

WeGive



2,073,840

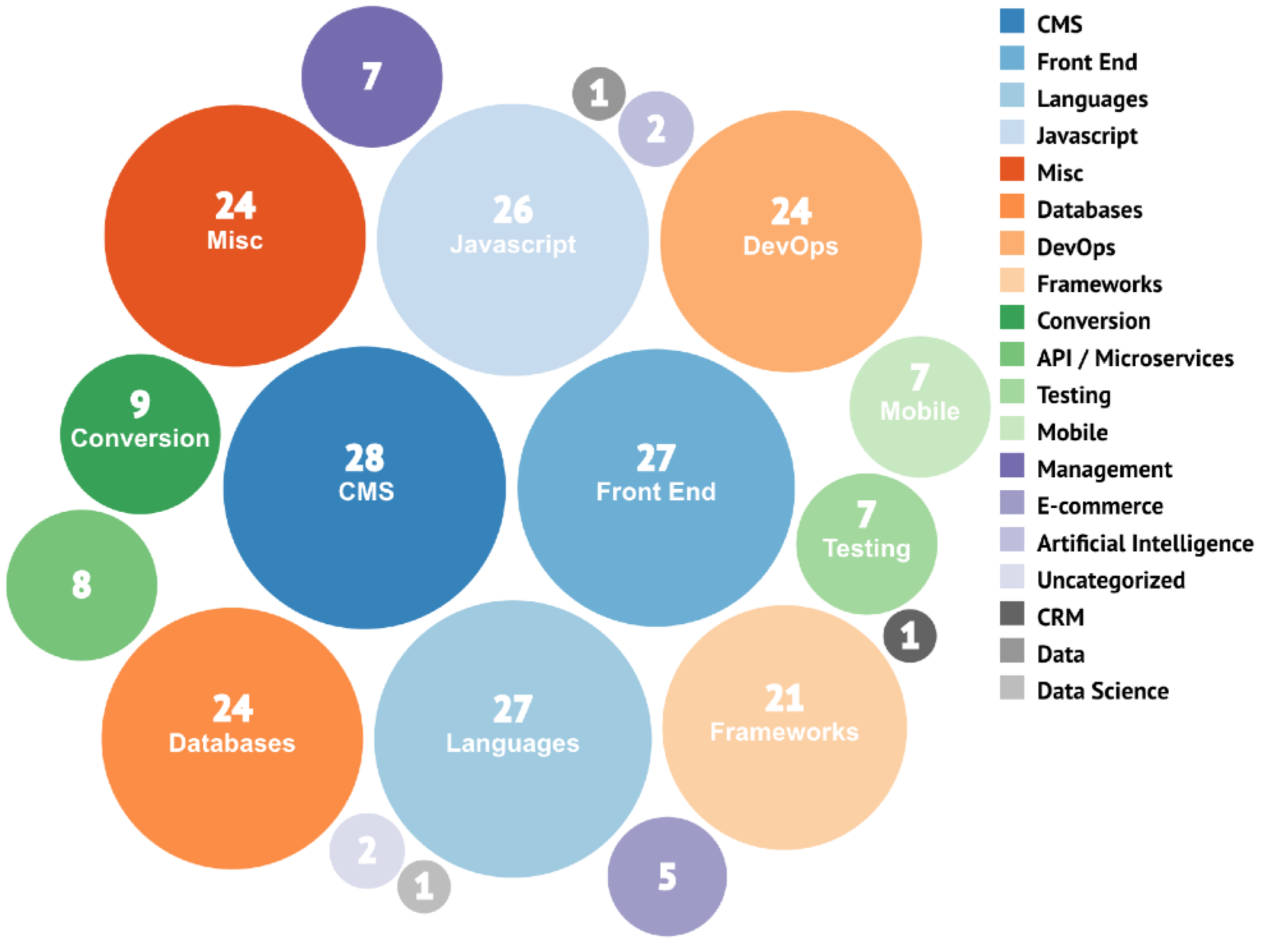
Downloads

weKnow

WeAre

weKnow

WeKnow



7
1
2
24
Misc
26
Javascript
24
DevOps
7
Mobile
9
Conversion
28
CMS
27
Front End
7
Testing
8
1
24
Databases
27
Languages
21
Frameworks
2
1
5
CMS
Front End
Languages
Javascript
Misc
Databases
DevOps
Frameworks
Conversion
API / Microservices
Testing
Mobile
Management
E-commerce
Artificial Intelligence
Uncategorized
CRM
Data
Data Science

Symfony

API Platform / GraphQL

ReactJS / Redux / Saga

Ant Design

weKnow

Symfony Flex

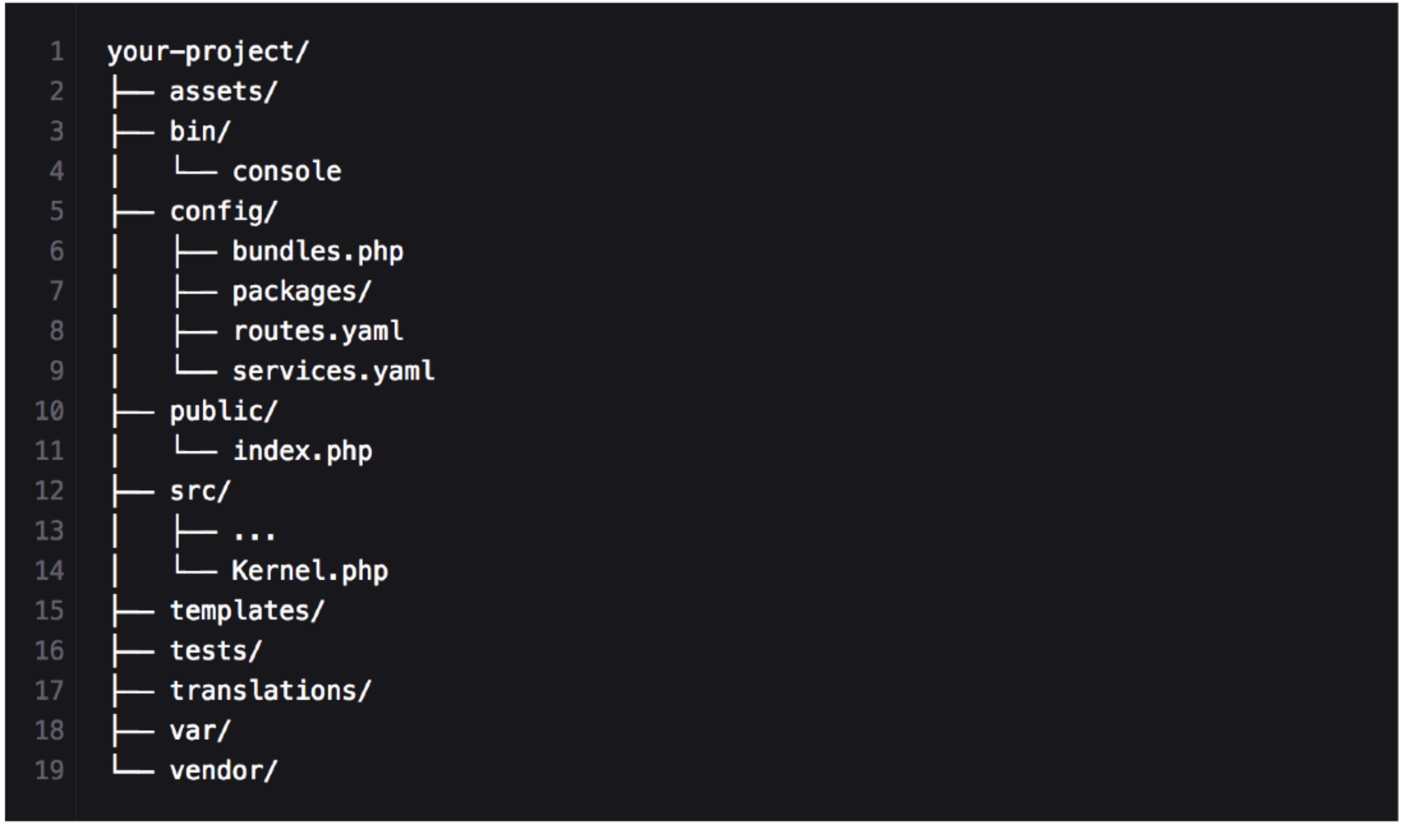
weKnow

Symfony Flex … a Composer plugin for Symfony

> Symfony Flex is a **Composer plugin** that modifies the behavior of the **require**, **update**, and **remove composer** commands.

> Symfony Flex automates the most common tasks of Symfony applications, like installing and removing bundles and other dependencies using recipes defined in a **manifest.json** file.

Directory structure

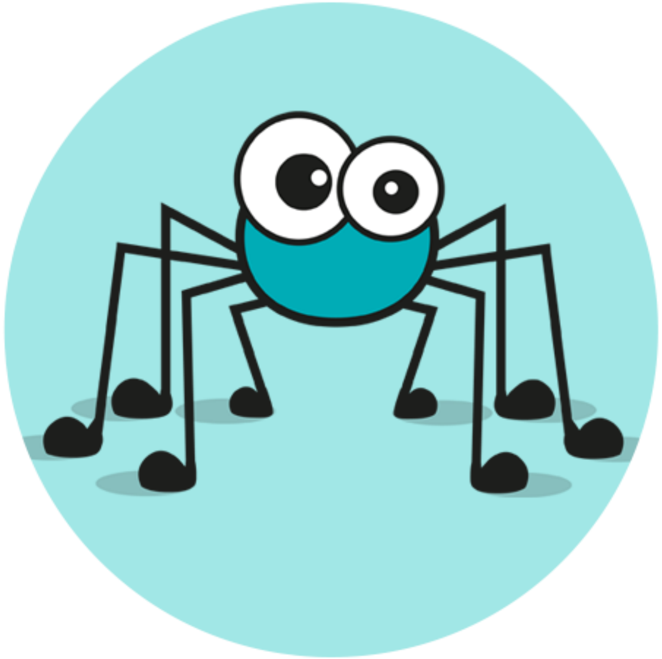


```
your-project/
├── assets/
├── bin/
│   └── console
├── config/
│   ├── bundles.php
│   ├── packages/
│   ├── routes.yaml
│   └── services.yaml
├── public/
│   └── index.php
├── src/
│   ├── ...
│   └── Kernel.php
├── templates/
├── tests/
├── translations/
├── var/
└── vendor/
```

API Platform Framework

weKnow

The API Platform Framework



REST and **GraphQL** framework to build modern API-driven projects

https://api-platform.com/

weKnow

Built on the Shoulders of Giants

> Extend the framework with thousands of existing Symfony bundles and React components.

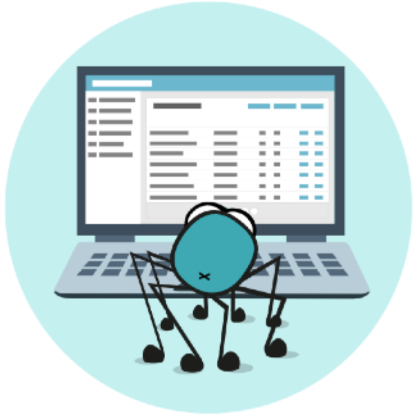
> The API component includes the **Symfony** 4 flex, the **Doctrine ORM**. Client-side components and **Admin** based on **React** and a **Docker** configuration ready to startup your project using one single command.

> Reuse all your **Symfony**, **React** and **Docker** skills and benefit of their high quality docs; you are in known territory.

The API Platform Components

API

Schema



Admin

CRUD

weKnow

Try API-Platform

```
# Clone code repository
git clone https://github.com/api-platform/api-platform.git
```

Recommendations and adjustments

> Update route prefix at **api/config/routes/api_platform.yaml** file.

```
api_platform:
    ...
    prefix: /api
```

> Update **admin/.env** and **client/.env** files (change protocol and port).

```
REACT_APP_API_ENTRYPOINT=http://localhost:8080/api
```

Start containers … and grab water, coffee, or a beer.

```
# Start containers

docker-compose up -d


# Open browser

open http://localhost/
```

THE #1 EVANGELIST EXCUSE FOR LEGITIMATELY SLACKING OFF:
"DOCKER'S PULLING IMAGES."
HEY! GET BACK TO WORK!
PULLING!
OH. CARRY ON.

weKnow

Add more formats

Update **api/config/packages/api_platform.yaml** adding:

```
formats:
  jsonld:  [‘application/ld+json'] # first one is the default format
  json:    ['application/json']
  jsonhal: ['application/hal+json']
  xml:     ['application/xml', 'text/xml']
  yaml:    ['application/x-yaml']
  csv:     ['text/csv']
  html:    ['text/html']
```

Add Blog and Tag entities, remove default Greeting entity

> Add new entities to **api/src/Entity/** directory:

api/src/Entity/Post.php

api/src/Entity/PostType.php

api/src/Entity/User.php

> Remove default entity

api/src/Entity/Greeting.php

api/src/Entity/Post.php 1/3

```
<?php
namespace App\Entity;

use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ApiResource
 * @ORM\Table(name="post")
 * @ORM\Entity
 */

class Post

{

…

}
```

api/src/Entity/Post.php 2/3

```
/**
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="AUTO")
 * @ORM\Column(type="integer")
 */
private $id;

/**
 * @ORM\Column
 * @Assert\NotBlank
 */
public $title = '';


/**
 * @ORM\Column

 * @Assert\NotBlank

 */

public $body = '';
```

api/src/Entity/Post.php 3/3

```
/**
 * @ORM\ManyToOne(targetEntity="PostType")
 * @ORM\JoinColumn(name="post_type_id", referencedColumnName="id", nullable=false)
 */
public $type;


public function getId(): int
{
    return $this->id;
}
```

Tracking Database changes

```
# Add dependency

composer require doctrine/doctrine-migrations-bundle



# Execute command(s)

doctrine:migrations:diff

doctrine:migrations:migrate
```

Add FOSUserBundle

```
# Add dependency

composer require friendsofsymfony/user-bundle

composer require symfony/swiftmailer-bundle
```

https://symfony.com/doc/current/bundles/FOSUserBundle/index.html

https://jolicode.com/blog/do-not-use-fosuserbundle

Initialize the project

> Drop and Create Database

> Execute Migrations

> Populate Entities with Data

```
bin/console init
```

NOTE: Use **hautelook/AliceBundle** or **willdurand/BazingaFakerBundle**

Loading Posts using the Browser

http://localhost:8080/**api/posts**

http://localhost:8080/**api/posts.json**

http://localhost:8080/**api/posts.jsonld**

http://localhost:8080/**api/posts/1**

http://localhost:8080/**api/posts/1.json**

Loading Posts using the CLI

```
curl -X GET "http://localhost:8080/api/posts" \
-H "accept: application/json"


curl -X GET "http://localhost:8080/api/posts/1" \
-H "accept: application/ld+json"
```

ADD Posts from CLI

```
curl -X POST "http://localhost:8080/api/posts" \
-H "accept: application/ld+json" \
-H "Content-Type: application/ld+json" \
-d '{ "title": "Post create from CLI", "body": "body-
less", "type": "/api/post_types/1"}'
```

UPDATE and REMOVE Posts from CLI

```
curl -X PUT "http://localhost:8080/api/posts/9" \
-H "accept: application/ld+json" \
-H "Content-Type: application/ld+json" \
-d '{ "title": "Updated from CLI"}'


curl -X DELETE "http://localhost:8080/api/posts/10" \
-H "accept: application/json"
```

Serialization

> API Platform allows to specify the which attributes of the resource are exposed during the normalization (**read**) and denormalization (**write**) process. It relies on the serialization (and deserialization) groups feature of the **Symfony Serializer** component.

> In addition to groups, you can use any option supported by the Symfony Serializer such as **enable_max_depth** to limit the serialization depth.

Serialization Relations (Post => PostType) 1/2

```
# api/src/Entity/Post.php & PostType.php

* @ApiResource(attributes={

*       "normalization_context"={"groups"={"read"}},

*       "denormalization_context"={"groups"={"write"}}

* })
```

Serialization Relations (Post => PostType) 2/2

```
# Add use keyword to class

use Symfony\Component\Serializer\Annotation\Groups;



# Add use keyword to properties

* @Groups({"read"})

* @Groups({"read", "write"})
```

GraphQL

weKnow

GraphQL

GraphQL offers significantly more flexibility for integrators.

Allows you to define in detail the only the data you want.

GraphQL lets you replace multiple REST requests with a single call to fetch the data you specify.

Add GraphQL

To enable **GraphQL** and **GraphiQL** interface in your API, simply require the **graphql-php** package using **Composer**:

```
composer require webonyx/graphql-php
```

```
open http://localhost:8080/api/graphql
```

Disable GraphiQL

Update **api/config/packages/api_platform.yaml** adding:

```
api_platform:
# ...
    graphql:
        graphiql:
            enabled: false
# ...
```

Load resource using GraphQL

```
{
  post (id:"/api/posts/1") {
    id,
    title,
    body
  }
}
```

Load resource using GraphQL form the CLI

```
curl -X POST \

-H "Content-Type: application/json" \

-d '{ "query": "{ post(id:\"/api/posts/1\") { id,
title, body }}" }' \

http://localhost:8080/api/graphql
```

Load resource relations using GraphQL

```
{
    post (id:"/api/posts/1") {
    title,
    body,
    type {
      id,
      name,
      machineName
    }
  }
}
```

Load resource relations using GraphQL form the CLI

```
curl -X POST \

-H "Content-Type: application/json" \

-d '{ "query": "{ post(id:\"/api/posts/1\") { id, title,
body, type { id, name, machineName } }}" }' \

http://localhost:8080/api/graphql
```

JWT

weKnow

JWT Dependencies

```
# JWT

composer require lexik/jwt-authentication-bundle



JWT Refresh
gesdinet/jwt-refresh-token-bundle
```

JWT Events (create)

```
# config/services.yaml
App\EventListener\JWTCreatedListener:
  tags:
    - {
        name: kernel.event_listener,
        event: lexik_jwt_authentication.on_jwt_created,
        method: onJWTCreated
      }


# src/EventListener/JWTCreatedListener.php
public function onJWTCreated(JWTCreatedEvent $event)
{
  $data = $event->getData();
  $user = $event->getUser();
  $data['organization'] = $user->getOrganization()->getId();
  $event->setData($data);
}
```

JWT Events (success)

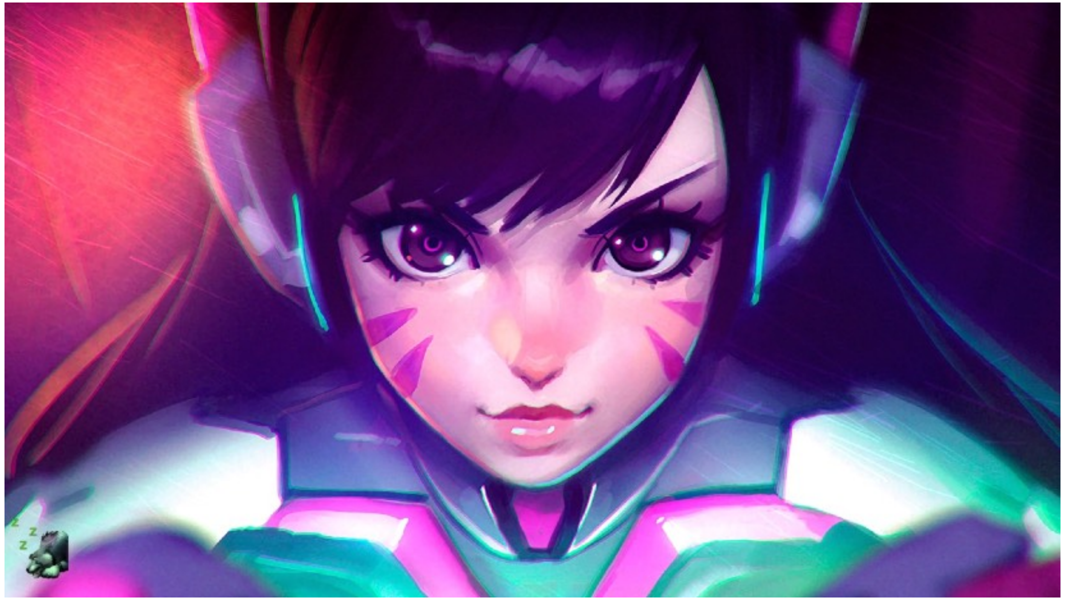
```
# config/services.yaml
App\EventListener\AuthenticationSuccessListener:
  tags:
    - {
        name: kernel.event_listener,
        event: lexik_jwt_authentication.on_authentication_success,
        method: onAuthenticationSuccessResponse
      }


# src/EventListener/AuthenticationSuccessListener.php
public function onAuthenticationSuccessResponse(AuthenticationSuccessEvent $event)
{
  $data = $event->getData();
  $user = $event->getUser();
  $data[‘roles'] = $user->getOrganization()->getRoles();
  $event->setData($data);
}
```

React+Redux+Saga+ AntDesign

weKnow

dvajs/dva - React and redux based framework.



https://github.com/dvajs/dva

React / Redux / Saga / AntDesig

Tips

> Use **webpack** instead of **roadhog**.

> Use **apollo-fetch** for **GraphQL** calls.

> Use **jwt-decode** to interact with **JWT**.

> Use **LocalStorage** for simple key/values storage.

> Use **IndexedDB** for encrypted and/or more complex data structures.

> Use **Socket-IO** + **Redis** to sync **API** with **ReactJS** client.

Directory Structure

```
├── package.json
├── src
│    ├── components
│    ├── constants.js
│    ├── index.js
│    ├── models
│    ├── router.js
│    ├── routes
│    └── services
└── webpack.config.js
```

constants.js

```
export const API_URL = process.env.API_ENTRY_POINT;

export const GRAPHQL_URL = `${API_URL}/api/graphql`;

export const NOTIFICATION_URL = process.env.NOTIFICATION_URL;

export const NOTIFICATION_ICON = {

    info: 'info',

    warning: 'exclamation',

    error: 'close',

    success: 'check'

};
```

index.js

```
import dva from 'dva';
import auth from './models/auth';
import local from './models/local';
import ui from './models/ui';

const app = dva({
  …
});

# Register global models
app.model(auth);
app.model(local);
app.model(ui);


app.router(require('./router'));
app.start(‘#root');
```

router.js 1/2

```
import …
import AuthRoute from './components/Auth/AuthRoute';

export default function RouterConfig({ history, app }) {
    const Login = dynamic({
        app,
        component: () => import('./routes/Login'),
    });

    const routes = [{
        path: '/posts',
        models: () => [
            import('./models/posts')
        ],
        component: () => import('./routes/Posts'),
    }, {

        path: '/posts/:id',
        models: () => [
            import('./models/projects'),
        ],
        component: () => import('./routes/Posts'),
    }];
```

router.js 2/2

```
    return (

        <Router history={history}>
            <Switch>
                <Route exact path="/" render={() => (<Redirect to="/login" />)} />
                <Route exact path="/login" component={Login} />
                {
                    routes.map(({ path, onEnter, ...dynamics }, key) => (

                        <AuthRoute
                            key={key} exact path={path}
                            component={dynamic({
                                app,
                                ...dynamics,
                            })}
                        />
                    ))

                }
            </Switch>
        </Router>
    );
}
```

src/components/Auth/AuthRoute.js

```
import {Route, Redirect} from "dva/router";
…

class AuthRoute extends Route {

  async isTokenValid() {
   # Check for token and refresh if not valid
  }

  render() {

    return (

        <Async

            promise={this.isTokenValid()}

            then={(isValid) => isValid ?

                <Route path={this.props.path} component={this.props.component} />

                :

                <Redirect to={{ pathname: '/login' }}/>

            }

        />

    );

};
```

src/models/posts.js 1/3

```
import {Route, Redirect} from "dva/router";
import * as postService from '../services/base';
import _map from 'lodash/map';
…

export default {
    namespace: ‘posts',
    state: {
        list: [],
        total: 0
    },
    reducers: {
        save(state, { payload: { list, total } }) {
            return { ...state, list, total};
        },

    },
```

src/models/posts.js 2/3

```
effects: {
    *fetch({ payload: { … }, { call, put }) {
        let { list, total } = yield select(state => state.posts);
        const graphQuery = `{
        posts(first: 10) {
          edges {
            node {
              id, _id, title, body,
              type {
                id, name, machineName
              }
            }
          }
        }
      }`;
```

src/models/posts.js 2/3

```
#  Async call with Promise support.
const posts = yield call(postService.fetch, 'posts', { … });
const list = _map(posts.data.posts.edges, posts => posts.node)
# Action triggering.
yield put({
    type: 'save',
    payload: {
        list,
        total,
    },
});
```

Directory Structure

```
src/
├── components
    ├── Auth
    ├── Generic
    ├── Layout
    └── Posts
        ├── ProjectBase.js
        ├── ProjectEdit.js
        ├── ProjectList.js
        └── ProjectNew.js
```

Any questions?

wk

weknowinc.com