Pioneer Project

---Improving yourself continuously and you are seeing the light !

How to show code in two screens in vscode: cmd+shift+p type duplicate workspace …

1 Write home page with basic react and sass CSS:

1-1 sass is a package of npm and we can use it to connect sass files and react project.

1-2 you can use background for img when you need to adjust the image.

1-3 Attention where is the home directory when you write the path. It can be different. Sass files and src in html are totally different.

1-4: header responsive CSS revisit:

We want the navigation bar occpy 40% of the page width, but only more than 40% when it is shrink to impact the max-content:

In .header:

grid-template-columns: minmax(max-content, 60vw) minmax(40vw, max-content);

@media only screen and (max-width: $bp-largest) {

grid-template-columns: minmax(max-content, 40vw) minmax(60vw, max-content);

}

Then in &\_\_nav mentioning:

justify-self: self-end;

2 Configure Postgres DB in k8s environment:

<https://arctype.com/blog/deploy-postgres-kubernetes/>

2-1 We are using helm: the package manager for k8s.

Helm: Chart - Contains Information required to create an application instance in Kubernetes

Config - Contains the configuration information that can be merged with a chart to create a releasable object.

Release - A running instance of a chart, combined with a specific config.

2-2 we can use homebrew to install helm. Using existing helm charts is easier instead of building a service from scratch. We can search Artifact Hub --- official distributed community helm chart repo. <https://artifacthub.io/>

2-3 The helm install command will make the deployment using chart happen:

helm install postgres-pioneer --set auth.postgresPassword=cfan bitnami/postgresql

Note: We have set the default password for user postgres above. All the set parameters please refer to its document.

When you are using chart, you can avoid writing a lot of different yaml files to config the DB. (the chart will help you build it and it is ready to use immediately.

And you can get the all API for that deployment:

NAME: postgres-pioneer

LAST DEPLOYED: Fri Sep 30 14:26:57 2022

NAMESPACE: default

STATUS: deployed

REVISION: 1

TEST SUITE: None

NOTES:

CHART NAME: postgresql

CHART VERSION: 11.9.2

APP VERSION: 14.5.0

\*\* Please be patient while the chart is being deployed \*\*

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

postgres-pioneer-postgresql.default.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

export POSTGRES\_PASSWORD=$(kubectl get secret --namespace default postgres-pioneer-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)

To connect to your database run the following command:

kubectl run postgres-pioneer-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:14.5.0-debian-11-r14 --env="PGPASSWORD=$POSTGRES\_PASSWORD" \

--command -- psql --host postgres-pioneer-postgresql -U postgres -d postgres -p 5432

> NOTE: If you access the container using bash, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in order to avoid the error "psql: local user with ID 1001} does not exist"

To connect to your database from outside the cluster execute the following commands:

kubectl port-forward --namespace default svc/postgres-pioneer-postgresql 5432:5432 &

PGPASSWORD="$POSTGRES\_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres -p 5432

You can use the command above to double verify the default password is set as cfan for the default username postgres.

Then you should set a port-forward from your local to k8s cluster postgres DB instance:

kubectl port-forward --namespace default svc/postgres-pioneer-postgresql 5432:5432

Please see the service name is: postgres-pioneer-postgresql.

Then you can connect from pgadmin. Remember to enter password as cfan:

Graphical user interface, application

Description automatically generated

2-4 We want to run migrate to set up the DB. To make the connect. You can provide DB connection url as an argument of the npm command.

DATABASE\_URL=postgres://<username>:<password>@localhost:5432/<dbname> npm run migrate up

So we fill it as:

DATABASE\_URL=postgres://postgres:cfan@localhost:5432/pioneerDB npm run migrate up

The command should get run in the top of migration folder.

2-5: Remember in the k8s cluster, each pod has its name and you can use its name as the hostname and try to call it.

2-6: Again please note when directing the routes, ingress-nginx does not care the port number from your request.

3 React Router.

3-1 We are getting react router from modern react with redux course. It is using version 5. Note that the version 6 will totally break the code so make sure you use a specific version to install.

npm install react-router-dom@5

3-2 Using skaffold and it will be slow to see the changes. So I will not use k8s environment when developing. The DB is in k8s so I set a port-forward. For client communications with query service, I set a proxy using setupProxy.js in create-react-app.

3-3 Note that setupProxy.js does not work in Typescript version. So we keep it as a js file instead of ts file. Then in the tsconfig, we exclude this only file from ts (as an exception):

Part of tsconfig.json:

"include": [

"src"

],

"exclude": ["src/setupProxy.js"]

4 Implement redux using ts:

4-1 Using TS to make request. Here is the format to parse a data with specific type:

const { data }: { data: Blog } = await buildClient().get(`/query/blog/${id}`);

The first bracket is to parse the data, the second bracket is to declare the type(s) inside.

4-2 reduxForm:

When I code, I found the redux form is working well if it is in a class based component but not work well if it is a functional component. Try some changes if you are stuck. Class based component may be better as it is extending the react component and can get a lot of helper methods. But a functional component lacks of that and the redux form may not work well with it.

Sample reduxForm:

import React from 'react';

import { Field, reduxForm, InjectedFormProps, WrappedFieldProps, WrappedFieldMetaProps } from 'redux-form';

import { Blog } from '../../state/actions';

interface FormFieldProps extends WrappedFieldProps {

label: string;

}

interface BlogFormInheritedProps {

onSubmit: (formValue: Blog) => void;

}

class BlogForm extends React.Component<BlogFormInheritedProps & InjectedFormProps<Blog, BlogFormInheritedProps>> {

renderError = ({ error, touched }: WrappedFieldMetaProps) => {

if (touched && error) {

return <div className="blog\_\_error">

<div className="error">{error}</div>

</div>

}

}

renderInput = ({input, label, meta}: FormFieldProps) => {

console.log(input)

const errorClassName = `${meta.error && meta.touched ? 'blog\_\_error' : ''}`;

return (

<div className="blog\_\_field">

<label className="blog\_\_label">{label}</label>

<input className="blog\_\_input" autoComplete="off" {...input} />

<div className={errorClassName}>{this.renderError(meta)}</div>

</div>

);

}

renderText = ({ input, label, meta }: FormFieldProps) => {

const errorClassName = `${meta.error && meta.touched ? 'blog\_\_error' : ''}`;

return (

<div className="blog\_\_field">

<label className="blog\_\_label">{label}</label>

<textarea className="blog\_\_text" autoComplete='off' {...input}>

</textarea>

<div className={errorClassName}>{this.renderError(meta)}</div>

</div>

);

}

render() {

const { onSubmit } = this.props;

return (

<form onSubmit={this.props.handleSubmit(onSubmit)} className="blog\_\_form">

<Field name="title" component={this.renderInput} label="Title" />

<Field name="description" component={this.renderInput} label="Description" />

<Field name="contents" component={this.renderText} label="Blog Contents" />

<button className="btn blog\_\_btn">Submit</button>

</form>

);

}

}

const validate = (formValues: Blog) => {

const errors: { title?: string, description?: string } = {};

if(!formValues.title) {

errors.title = 'You must enter a title';

}

if(!formValues.description) {

errors.description = 'You must enter a description';

}

return errors;

}

const formWrapped = reduxForm<Blog, BlogFormInheritedProps>({

form: 'blogForm',

validate: validate,

})(BlogForm);

export default formWrapped;

Note in the code above, how we combine the InjectedFormProps and the props getting from the parent component (BlogFormInheritedProps)

First, in the class component generic type field, write both of them. They are connected with &, and the inherited props show as a generic type of the InjectedFormProps again.

class BlogForm extends React.Component<BlogFormInheritedProps & InjectedFormProps<Blog, BlogFormInheritedProps>> {

Second, in the final reduxForm HOF we need to add both props are the generic type field:

const formWrapped = reduxForm<Blog, BlogFormInheritedProps>({

form: 'blogForm',

validate: validate,

})(BlogForm);

4-3 textarea and input will automatically records user’s line break. To show it onscreen with line break you need to use the following css:

white-space: pre-line;

4-4 While react is going from react 17 to react 18, it is not compatible with react router dom @5 any more and we were doing an upgrade from 5 to 6. A lot of issues are coming and I have googled to resolve them. The new HistoryRouter is there for us to use in case we want the redirect happen outside the react component, such as in actionCreator.

4-5: When there are dependencies compatible issues, you can run as

npm install --legacy-peer-deps

4-6: When you see CreateContainerConfigError, please note if the container needs a secret or config file in config file and it needs to be added.

4-7: While you are working on different projects using ingress-nginx you need to reset it by using the command:

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.0/deploy/static/provider/cloud/deploy.yaml

4-8: If one repo was working before but not working after latest clone, the reason may be:

4-8-1: Some request links need to be configured well based om the environment, such as the baseURL for axios:

const buildClient = ({ req }) => {

if (typeof window === 'undefined') {

//we are on the server

return axios.create({

baseURL:

'http://ingress-nginx-controller.ingress-nginx.svc.cluster.local',

headers: req.headers,

});

} else {

//we are on the browser

return axios.create({

baseUrl: '/', //No need for baseURL or header config as browser will auto handle it

});

}

};

4-8-2: Make sue ingress-nginx pod is reset per the 4-7 command.

4-8-3: Check the error to decide the problem, such as latest node version, or dependencies issue, try to update Dockerfile to fix it.

4-9: