Redux

Payload:

import {createStore, applyMiddleware} from 'redux';

import logger from 'redux-logger';

//store

const store = createStore(reducer, applyMiddleware(logger.default));

const history = [];

function reducer(state={amount:1}, action){

if(action.type==='increment'){

//mutable

//state.amount=state.amount+1;

//immutable

return {amount: state.amount + action.payload}

}

return state

}

//global state

// store.subscribe(()=>{

// history.push(store.getState())

// console.log(history);

// })

setInterval(()=>{

store.dispatch({type:'increment', payload:4})

},3000)

Action Creator:

import {createStore, applyMiddleware} from 'redux';

import logger from 'redux-logger';

//store

const store = createStore(reducer, applyMiddleware(logger.default));

const history = [];

function reducer(state={amount:1}, action){

if(action.type==='increment'){

//mutable

//state.amount=state.amount+1;

//immutable

return {amount: state.amount + action.payload}

}

return state

}

//action creator

function increment(){

return {type:'increment', payload:4}

}

setInterval(()=>{

store.dispatch(increment())

},3000)

Make string const:

import {createStore, applyMiddleware} from 'redux';

import logger from 'redux-logger';

//store

const history = [];

const inc= 'increment';

const store = createStore(reducer, applyMiddleware(logger.default));

function reducer(state={amount:1}, action){

if(action.type===inc){

return {amount: state.amount + action.payload}

}

return state

}

//action creator

function increment(value){

return {type:inc, payload:value}

}

setInterval(()=>{

store.dispatch(increment(3))

},3000)

**Create fake json-server:**

Run command: npm I -g json-server

Or sudo npm I -g json-server

Then create file : db.json

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | { | |  | "accounts": [ | |  | { "id": 1, "amount": 200 }, | |  | { "id": 2, "amount": 100 } | |  | ], | |  | "bonus": [ | |  | { "id": 1, "points": 2 }, | |  | { "id": 2, "points": 3 } | |  | ] | |  | } | |

Run this server using command: json-server db.json

Now Install Axios: **npm install axios**

Install Redux-thunk: **npm install redux-thunk**

**Update Index.js**

|  |
| --- |
| import { createStore, applyMiddleware, combineReducers } from 'redux';  import logger from 'redux-logger';  import thunk from 'redux-thunk';  import axios from 'axios';  //action name constants  // const init = 'account/init';  const inc = 'account/increment';  const dec = 'account/decrement';  const incByAmt = 'account/incrementByAmount';  const getAccUserPending = 'account/getUser/pending';  const getAccUserFulFilled = 'account/getUser/fulfilled';  const getAccUserRejected = 'account/getUser/rejected';  const incBonus = 'bonus/increment';  //store  const store = createStore(  combineReducers({  account: accountReducer,  bonus: bonusReducer  }),  applyMiddleware(logger.default, thunk.default)  );  const history = [];  //reducer  function accountReducer(state = { amount: 1 }, action) {  switch (action.type) {  case getAccUserFulFilled:  return { amount: action.payload, pending:false };  case getAccUserRejected:  return {...state, error:action.error, pending:false };  case getAccUserPending:  return { ...state,pending:true };  case inc:  return { amount: state.amount + 1 };  case dec:  return { amount: state.amount - 1 };  case incByAmt:  return { amount: state.amount + action.payload };  default:  return state;  }  }  function bonusReducer(state = { points: 0 }, action) {  switch (action.type) {  case incBonus:  return { points: state.points + 1 };  case incByAmt:  if(action.payload>=100)  return { points: state.points + 1 };  default:  return state;  }  }  //Action creators  function getUserAccount(id) {  return async (dispatch, getState) => {  try{  dispatch(getAccountUserPending());  const { data } = await axios.get(`http://localhost:3000/accounts/${id}`);  dispatch(getAccountUserFulFilled(data.amount));  } catch(error){  dispatch(getAccountUserRejected(error.message));  }    };  }  function getAccountUserFulFilled(value) {  return { type: getAccUserFulFilled, payload: value };  }  function getAccountUserRejected(error) {  return { type: getAccUserRejected, error: error };  }  function getAccountUserPending() {  return { type: getAccUserPending };  }  function increment() {  return { type: inc };  }  function decrement() {  return { type: dec };  }  function incrementByAmount(value) {  return { type: incByAmt, payload: value };  }  function incrementBonus(value) {  return { type: incBonus};  }  setTimeout(() => {  store.dispatch(getUserAccount(2));  }, 2000); |