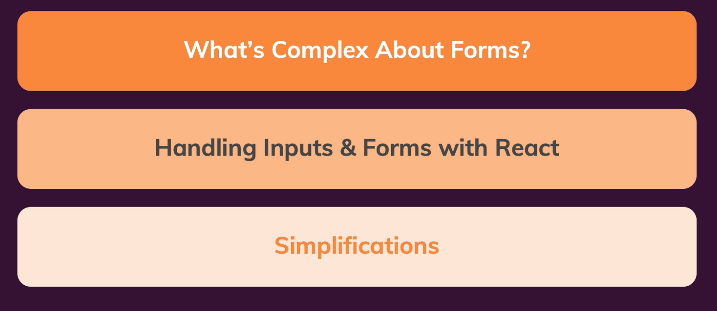
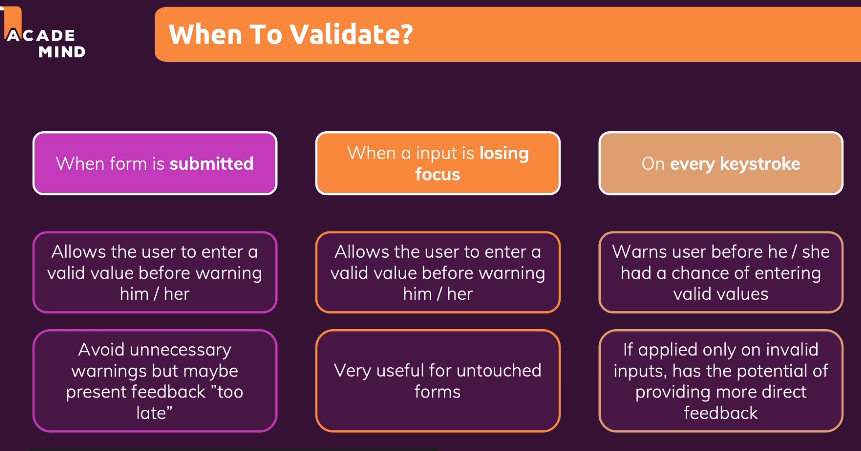
Section 16 Working With Forms & User Input



198. Our Starting Setup

Source to start: <https://github.com/academind/react-complete-guide-code/tree/16-working-with-forms>

199. What’s So Complex About Forms



200. Dealing with Form Submission & Getting user Input Values to lecture 208

Step 1: in SimpleInput.js

import { useState } from 'react';

const SimpleInput = (props) => {

  const [enteredName, setEnteredName] = useState('');

  const [enteredNameTouched, setEnteredNameTouched] = useState(false);

  const [enteredEmail, setEnteredEmail] = useState('');

  const [enteredEmailTouched, setEnteredEmailTouched] = useState(false);

  const enteredNameIsValid = enteredName.trim() !== '';

  const nameInputIsInvalid = !enteredNameIsValid && enteredNameTouched;

  const enteredEmailIsValid = enteredEmail.includes('@');

  const enteredEmailIsInvalid = !enteredEmailIsValid && enteredEmailTouched;

  let formIsValid = false;

  if (enteredNameIsValid && enteredEmailIsValid) {

    formIsValid = true;

  }

  const nameInputChangeHandler = (event) => {

    setEnteredName(event.target.value);

  };

  const emailInputChangeHandler = (event) => {

    setEnteredEmail(event.target.value);

  };

  const nameInputBlurHandler = (event) => {

    setEnteredNameTouched(true);

  };

  const emailInputBlurHandler = (event) => {

    setEnteredEmailTouched(true);

  };

  const formSubmissionHandler = (event) => {

    event.preventDefault();

    setEnteredNameTouched(true);

    if (!enteredNameIsValid) {

      return;

    }

    console.log(enteredName);

    // nameInputRef.current.value = ''; => NOT IDEAL, DON'T MANIPULATE THE DOM

    setEnteredName('');

    setEnteredNameTouched(false);

    setEnteredEmail('');

    setEnteredEmailTouched(false);

  };

  const nameInputClasses = nameInputIsInvalid

    ? 'form-control invalid'

    : 'form-control';

  const emailInputClasses = enteredEmailIsInvalid

    ? 'form-control invalid'

    : 'form-control';

  return (

    <form onSubmit={formSubmissionHandler}>

      <div className={nameInputClasses}>

        <label htmlFor='name'>Your Name</label>

        <input

          type='text'

          id='name'

          onChange={nameInputChangeHandler}

          onBlur={nameInputBlurHandler}

          value={enteredName}

        />

        {nameInputIsInvalid && (

          <p className='error-text'>Name must not be empty.</p>

        )}

      </div>

      <div className={emailInputClasses}>

        <label htmlFor='email'>Your E-Mail</label>

        <input

          type='email'

          id='email'

          onChange={emailInputChangeHandler}

          onBlur={emailInputBlurHandler}

          value={enteredEmail}

        />

        {enteredEmailIsInvalid && (

          <p className='error-text'>Please enter a valid email.</p>

        )}

      </div>

      <div className='form-actions'>

        <button disabled={!formIsValid}>Submit</button>

      </div>

    </form>

  );

};

export default SimpleInput;

Step 2 in BasicForm.js

const BasicForm = (props) => {

  return (

    <form>

      <div className='control-group'>

        <div className='form-control'>

          <label htmlFor='name'>First Name</label>

          <input type='text' id='name' />

        </div>

        <div className='form-control'>

          <label htmlFor='name'>Last Name</label>

          <input type='text' id='name' />

        </div>

      </div>

      <div className='form-control'>

        <label htmlFor='name'>E-Mail Address</label>

        <input type='text' id='name' />

      </div>

      <div className='form-actions'>

        <button>Submit</button>

      </div>

    </form>

  );

};

export default BasicForm;

207. Adding A Custom Input Hook (use common input)

Step 1: create folder src/hooks/use-input.js

import { useState } from 'react';

const useInput = (validateValue) => {

  const [enteredValue, setEnteredValue] = useState('');

  const [isTouched, setIsTouched] = useState(false);

  const valueIsValid = validateValue(enteredValue);

  const hasError = !valueIsValid && isTouched;

  const valueChangeHandler = (event) => {

    setEnteredValue(event.target.value);

  };

  const inputBlurHandler = (event) => {

    setIsTouched(true);

  };

  const reset = () => {

    setEnteredValue('');

    setIsTouched(false);

  };

  return {

    value: enteredValue,

    isValid: valueIsValid,

    hasError,

    valueChangeHandler,

    inputBlurHandler,

    reset

  };

};

export default useInput;

Step 2: then in SimpleInput.js change like this

import { useState } from 'react';

import useInput from '../hooks/use-input';

const SimpleInput = (props) => {

  const {

    value: enteredName,

    isValid: enteredNameIsValid,

    hasError: nameInputHasError,

    valueChangeHandler: nameChangedHandler,

    inputBlurHandler: nameBlurHandler,

    reset: resetNameInput

  } = useInput(value => value.trim() !== '');

  const [enteredEmail, setEnteredEmail] = useState('');

  const [enteredEmailTouched, setEnteredEmailTouched] = useState(false);

  const enteredEmailIsValid = enteredEmail.includes('@');

  const enteredEmailIsInvalid = !enteredEmailIsValid && enteredEmailTouched;

  let formIsValid = false;

  if (enteredNameIsValid && enteredEmailIsValid) {

    formIsValid = true;

  }

  const emailInputChangeHandler = (event) => {

    setEnteredEmail(event.target.value);

  };

  const emailInputBlurHandler = (event) => {

    setEnteredEmailTouched(true);

  };

  const formSubmissionHandler = (event) => {

    event.preventDefault();

    if (!enteredNameIsValid) {

      return;

    }

    console.log(enteredName);

    // nameInputRef.current.value = ''; => NOT IDEAL, DON'T MANIPULATE THE DOM

    resetNameInput();

    setEnteredEmail('');

    setEnteredEmailTouched(false);

  };

  const nameInputClasses = nameInputHasError

    ? 'form-control invalid'

    : 'form-control';

  const emailInputClasses = enteredEmailIsInvalid

    ? 'form-control invalid'

    : 'form-control';

  return (

    <form onSubmit={formSubmissionHandler}>

      <div className={nameInputClasses}>

        <label htmlFor='name'>Your Name</label>

        <input

          type='text'

          id='name'

          onChange={nameChangedHandler}

          onBlur={nameBlurHandler}

          value={enteredName}

        />

        {nameInputHasError && (

          <p className='error-text'>Name must not be empty.</p>

        )}

      </div>

      <div className={emailInputClasses}>

        <label htmlFor='email'>Your E-Mail</label>

        <input

          type='email'

          id='email'

          onChange={emailInputChangeHandler}

          onBlur={emailInputBlurHandler}

          value={enteredEmail}

        />

        {enteredEmailIsInvalid && (

          <p className='error-text'>Please enter a valid email.</p>

        )}

      </div>

      <div className='form-actions'>

        <button disabled={!formIsValid}>Submit</button>

      </div>

    </form>

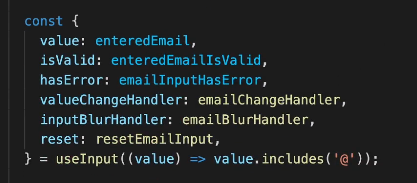
  );

};

export default SimpleInput;

208. Re-Using The Custom Hook

Step 1: Change SimpleInput.js



import useInput from '../hooks/use-input';

const SimpleInput = (props) => {

  const {

    value: enteredName,

    isValid: enteredNameIsValid,

    hasError: nameInputHasError,

    valueChangeHandler: nameChangedHandler,

    inputBlurHandler: nameBlurHandler,

    reset: resetNameInput,

  } = useInput((value) => value.trim() !== '');

  const {

    value: enteredEmail,

    isValid: enteredEmailIsValid,

    hasError: emailInputHasError,

    valueChangeHandler: emailChangeHandler,

    inputBlurHandler: emailBlurHandler,

    reset: resetEmailInput,

  } = useInput((value) => value.includes('@'));

  let formIsValid = false;

  if (enteredNameIsValid && enteredEmailIsValid) {

    formIsValid = true;

  }

  const formSubmissionHandler = (event) => {

    event.preventDefault();

    if (!enteredNameIsValid) {

      return;

    }

    console.log(enteredName);

    // nameInputRef.current.value = ''; => NOT IDEAL, DON'T MANIPULATE THE DOM

    resetNameInput();

    resetEmailInput();

  };

  const nameInputClasses = nameInputHasError

    ? 'form-control invalid'

    : 'form-control';

  const emailInputClasses = emailInputHasError

    ? 'form-control invalid'

    : 'form-control';

  return (

    <form onSubmit={formSubmissionHandler}>

      <div className={nameInputClasses}>

        <label htmlFor='name'>Your Name</label>

        <input

          type='text'

          id='name'

          onChange={nameChangedHandler}

          onBlur={nameBlurHandler}

          value={enteredName}

        />

        {nameInputHasError && (

          <p className='error-text'>Name must not be empty.</p>

        )}

      </div>

      <div className={emailInputClasses}>

        <label htmlFor='email'>Your E-Mail</label>

        <input

          type='email'

          id='email'

          onChange={emailChangeHandler}

          onBlur={emailBlurHandler}

          value={enteredEmail}

        />

        {emailInputHasError && (

          <p className='error-text'>Please enter a valid email.</p>

        )}

      </div>

      <div className='form-actions'>

        <button disabled={!formIsValid}>Submit</button>

      </div>

    </form>

  );

};

export default SimpleInput;

209. A Challenge For You

Build again from Section 16 to Lecture 209 (Practice)

210. Applying Our Hook & Knowledge To A New Form

Step 1: In BasicForm.js

import useInput from '../hooks/use-input';

const isNotEmpty = (value) => value.trim() !== '';

const isEmail = (value) => value.includes('@');

const BasicForm = (props) => {

  const {

    value: firstNameValue,

    isValid: firstNameIsValid,

    hasError: firstNameHasError,

    valueChangeHandler: firstNameChangeHandler,

    inputBlurHandler: firstNameBlurHandler,

    reset: resetFirstName,

  } = useInput(isNotEmpty);

  const {

    value: lastNameValue,

    isValid: lastNameIsValid,

    hasError: lastNameHasError,

    valueChangeHandler: lastNameChangeHandler,

    inputBlurHandler: lastNameBlurHandler,

    reset: resetLastName,

  } = useInput(isNotEmpty);

  const {

    value: emailValue,

    isValid: emailIsValid,

    hasError: emailHasError,

    valueChangeHandler: emailChangeHandler,

    inputBlurHandler: emailBlurHandler,

    reset: resetEmail,

  } = useInput(isEmail);

  let formIsValid = false;

  if (firstNameIsValid && lastNameIsValid && emailIsValid) {

    formIsValid = true;

  }

  const submitHandler = event => {

    event.preventDefault();

    if (!formIsValid) {

      return;

    }

    console.log('Submitted!');

    console.log(firstNameValue, lastNameValue, emailValue);

    resetFirstName();

    resetLastName();

    resetEmail();

  };

  const firstNameClasses = firstNameHasError ? 'form-control invalid' : 'form-control';

  const lastNameClasses = lastNameHasError ? 'form-control invalid' : 'form-control';

  const emailClasses = emailHasError ? 'form-control invalid' : 'form-control';

  return (

    <form onSubmit={submitHandler}>

      <div className='control-group'>

        <div className={firstNameClasses}>

          <label htmlFor='name'>First Name</label>

          <input

            type='text'

            id='name'

            value={firstNameValue}

            onChange={firstNameChangeHandler}

            onBlur={firstNameBlurHandler}

          />

          {firstNameHasError && <p className="error-text">Please enter a first name.</p>}

        </div>

        <div className={lastNameClasses}>

          <label htmlFor='name'>Last Name</label>

          <input

            type='text'

            id='name'

            value={lastNameValue}

            onChange={lastNameChangeHandler}

            onBlur={lastNameBlurHandler}

          />

          {lastNameHasError && <p className="error-text">Please enter a last name.</p>}

        </div>

      </div>

      <div className={emailClasses}>

        <label htmlFor='name'>E-Mail Address</label>

        <input

          type='text'

          id='name'

          value={emailValue}

          onChange={emailChangeHandler}

          onBlur={emailBlurHandler}

        />

        {emailHasError && <p className="error-text">Please enter a valid email address.</p>}

      </div>

      <div className='form-actions'>

        <button disabled={!formIsValid}>Submit</button>

      </div>

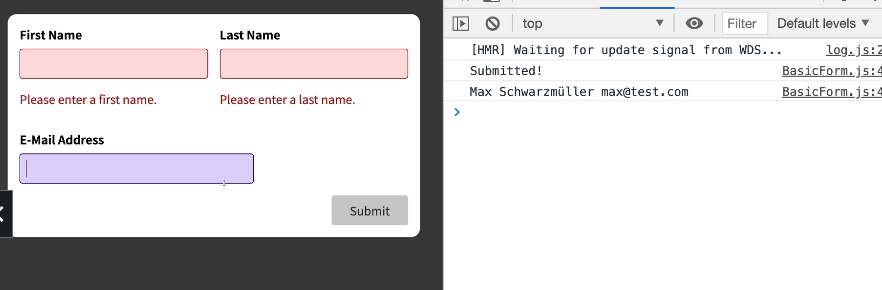
    </form>

  );

};

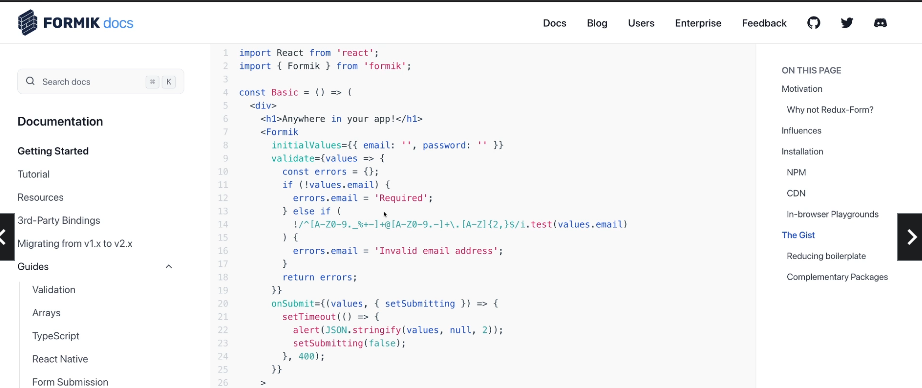
export default BasicForm;

Check result



211. Summary

Another solution when validate form



212. Bonus: Using useReducer()

Step 1: in use-input.js

import { useReducer } from 'react';

const initialInputState = {

  value: '',

  isTouched: false,

};

const inputStateReducer = (state, action) => {

  if (action.type === 'INPUT') {

    return { value: action.value, isTouched: state.isTouched };

  }

  if (action.type === 'BLUR') {

    return { isTouched: true, value: state.value };

  }

  if (action.type === 'RESET') {

    return { isTouched: false, value: '' };

  }

  return inputStateReducer;

};

const useInput = (validateValue) => {

  const [inputState, dispatch] = useReducer(

    inputStateReducer,

    initialInputState

  );

  const valueIsValid = validateValue(inputState.value);

  const hasError = !valueIsValid && inputState.isTouched;

  const valueChangeHandler = (event) => {

    dispatch({ type: 'INPUT', value: event.target.value });

  };

  const inputBlurHandler = (event) => {

    dispatch({ type: 'BLUR' });

  };

  const reset = () => {

    dispatch({ type: 'RESET' });

  };

  return {

    value: inputState.value,

    isValid: valueIsValid,

    hasError,

    valueChangeHandler,

    inputBlurHandler,

    reset,

  };

};

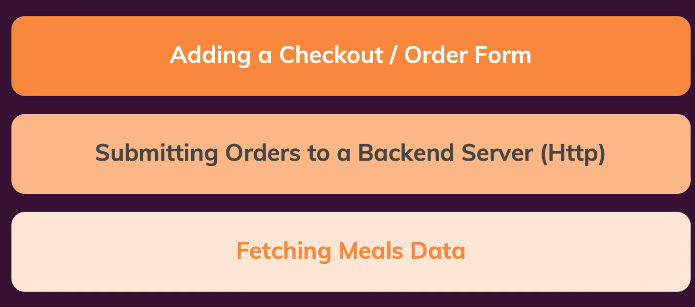
export default useInput;

213. Module Resources

You may want to **compare your code to mine** (e.g. to find + fix errors).

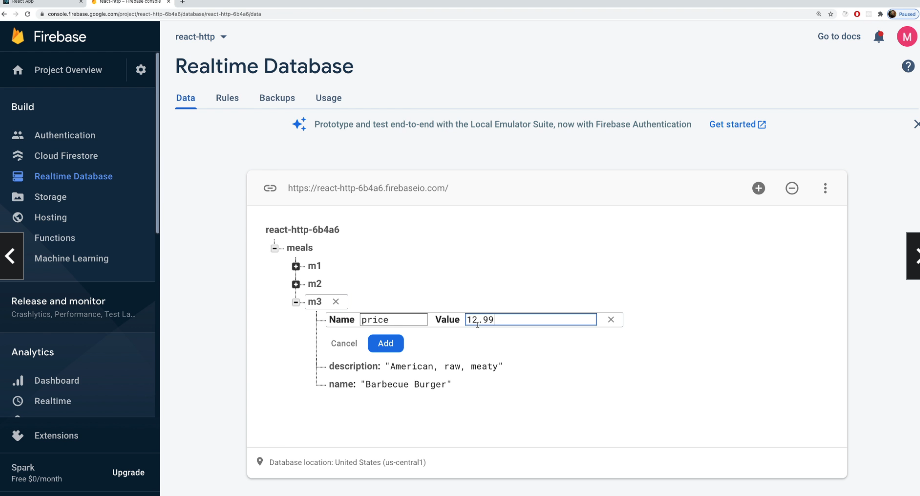
For that, you find **multiple code snapshots** for this module here in this Github repository: <https://github.com/academind/react-complete-guide-code/tree/16-working-with-forms>

Section 17: Practice Project Adding Http



215. Moving ‘Meals’ Data To The Backend

Step 1: in firebase create value



216. Fetching Meals via Http

Step 0: clone source from

https://github.com/academind/react-complete-guide-code/tree/17-practice-food-order-http-forms

Step 1: in AvailableMeals.js, add new api



import { useEffect, useState } from 'react';

import Card from '../UI/Card';

import MealItem from './MealItem/MealItem';

import classes from './AvailableMeals.module.css';

const AvailableMeals = () => {

  const [meals, setMeals] = useState([]);

  useEffect(() => {

    const fetchMeals = async () => {

      const response = await fetch('https://react-http-6b4a6.firebaseio.com/meals.json');

      const responseData = await response.json();

      const loadedMeals = [];

      for (const key in responseData) {

        loadedMeals.push({

          id: key,

          name: responseData[key].name,

          description: responseData[key].description,

          price: responseData[key].price,

        });

      }

      setMeals(loadedMeals);

    };

    fetchMeals();

  }, []);

  const mealsList = meals.map((meal) => (

    <MealItem

      key={meal.id}

      id={meal.id}

      name={meal.name}

      description={meal.description}

      price={meal.price}

    />

  ));

  return (

    <section className={classes.meals}>

      <Card>

        <ul>{mealsList}</ul>

      </Card>

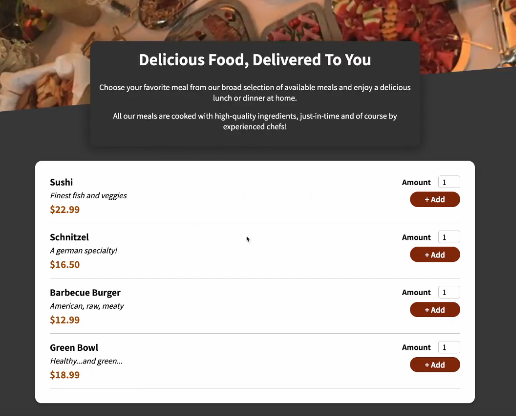
    </section>

  );

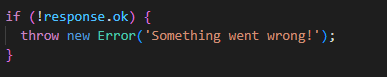
};

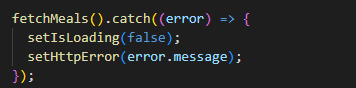
export default AvailableMeals;

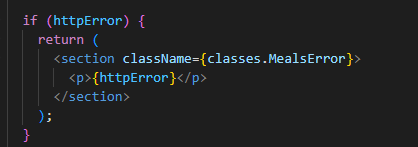
Step 2: check result



217. Handling The Loading State + 218 Handling Errors







Step 1: in AvailableMeals.js

import { useEffect, useState } from 'react';

import Card from '../UI/Card';

import MealItem from './MealItem/MealItem';

import classes from './AvailableMeals.module.css';

const AvailableMeals = () => {

  const [meals, setMeals] = useState([]);

  const [isLoading, setIsLoading] = useState(true);

  const [httpError, setHttpError] = useState();

  useEffect(() => {

    const fetchMeals = async () => {

      const response = await fetch(

        'https://react-http-6b4a6.firebaseio.com/meals.json'

      );

      if (!response.ok) {

        throw new Error('Something went wrong!');

      }

      const responseData = await response.json();

      const loadedMeals = [];

      for (const key in responseData) {

        loadedMeals.push({

          id: key,

          name: responseData[key].name,

          description: responseData[key].description,

          price: responseData[key].price,

        });

      }

      setMeals(loadedMeals);

      setIsLoading(false);

    };

    fetchMeals().catch((error) => {

      setIsLoading(false);

      setHttpError(error.message);

    });

  }, []);

  if (isLoading) {

    return (

      <section className={classes.MealsLoading}>

        <p>Loading...</p>

      </section>

    );

  }

  if (httpError) {

    return (

      <section className={classes.MealsError}>

        <p>{httpError}</p>

      </section>

    );

  }

  const mealsList = meals.map((meal) => (

    <MealItem

      key={meal.id}

      id={meal.id}

      name={meal.name}

      description={meal.description}

      price={meal.price}

    />

  ));

  return (

    <section className={classes.meals}>

      <Card>

        <ul>{mealsList}</ul>

      </Card>

    </section>

  );

};

export default AvailableMeals;

Step 2: in AvailableMeals.module.css

.meals {

  max-width: 60rem;

  width: 90%;

  margin: 2rem auto;

  animation: meals-appear 1s ease-out forwards;

}

.MealsLoading {

  text-align: center;

  color: white;

}

.MealsError {

  text-align: center;

  color: red;

}

.meals ul {

  list-style: none;

  margin: 0;

  padding: 0;

}

@keyframes meals-appear {

  from {

    opacity: 0;

    transform: translateY(3rem);

  }

  to {

    opacity: 1;

    transform: translateY(0);

  }

}

Step 3: check result and done with module Fetching Meals Data



219. Adding A Checkout Form

Step 1: create new components/Cart/Checkout.js

import classes from './Checkout.module.css';

const Checkout = (props) => {

  const confirmHandler = (event) => {

    event.preventDefault();

  };

  return (

    <form className={classes.form} onSubmit={confirmHandler}>

      <div className={classes.control}>

        <label htmlFor='name'>Your Name</label>

        <input type='text' id='name' />

      </div>

      <div className={classes.control}>

        <label htmlFor='street'>Street</label>

        <input type='text' id='street' />

      </div>

      <div className={classes.control}>

        <label htmlFor='postal'>Postal Code</label>

        <input type='text' id='postal' />

      </div>

      <div className={classes.control}>

        <label htmlFor='city'>City</label>

        <input type='text' id='city' />

      </div>

      <div className={classes.actions}>

        <button type='button' onClick={props.onCancel}>

          Cancel

        </button>

        <button className={classes.submit}>Confirm</button>

      </div>

    </form>

  );

};

export default Checkout;

Step 2: and also Checkout.module.css

.form {

  margin: 1rem 0;

  height: 17rem;

  overflow: auto;

}

.control {

  margin-bottom: 0.5rem;

}

.control label {

  font-weight: bold;

  margin-bottom: 0.25rem;

  display: block;

}

.control input {

  font: inherit;

  border: 1px solid #ccc;

  border-radius: 4px;

  width: 20rem;

  max-width: 100%;

}

.actions {

  display: flex;

  justify-content: flex-end;

  gap: 1rem;

}

.actions button {

  font: inherit;

  color: #5a1a01;

  cursor: pointer;

  background-color: transparent;

  border: none;

  border-radius: 25px;

  padding: 0.5rem 2rem;

}

.actions button:hover,

.actions button:active {

  background-color: #ffe6dc;

}

.actions .submit {

  border: 1px solid #5a1a01;

  background-color: #5a1a01;

  color: white;

}

.actions .submit:hover,

.actions .submit:active {

  background-color: #7a2706;

}

.invalid label {

  color: #ca3e51;

}

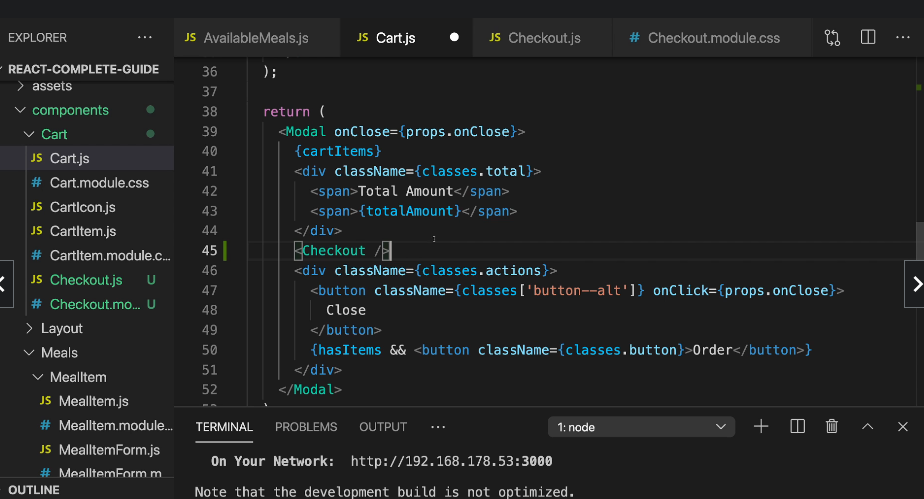
.invalid input {

  border-color: #aa0b20;

  background-color: #ffeff1;

}

Step 3 go to Cart.js add like this + and add onClick when submit



import { useContext, useState } from 'react';

import Modal from '../UI/Modal';

import CartItem from './CartItem';

import classes from './Cart.module.css';

import CartContext from '../../store/cart-context';

import Checkout from './Checkout';

const Cart = (props) => {

  const [isCheckout, setIsCheckout] = useState(false);

  const cartCtx = useContext(CartContext);

  const totalAmount = `$${cartCtx.totalAmount.toFixed(2)}`;

  const hasItems = cartCtx.items.length > 0;

  const cartItemRemoveHandler = (id) => {

    cartCtx.removeItem(id);

  };

  const cartItemAddHandler = (item) => {

    cartCtx.addItem(item);

  };

  const orderHandler = () => {

    setIsCheckout(true);

  };

  const cartItems = (

    <ul className={classes['cart-items']}>

      {cartCtx.items.map((item) => (

        <CartItem

          key={item.id}

          name={item.name}

          amount={item.amount}

          price={item.price}

          onRemove={cartItemRemoveHandler.bind(null, item.id)}

          onAdd={cartItemAddHandler.bind(null, item)}

        />

      ))}

    </ul>

  );

  const modalActions = (

    <div className={classes.actions}>

      <button className={classes['button--alt']} onClick={props.onClose}>

        Close

      </button>

      {hasItems && (

        <button className={classes.button} onClick={orderHandler}>

          Order

        </button>

      )}

    </div>

  );

  return (

    <Modal onClose={props.onClose}>

      {cartItems}

      <div className={classes.total}>

        <span>Total Amount</span>

        <span>{totalAmount}</span>

      </div>

      {isCheckout && <Checkout onCancel={props.onClose} />}

      {!isCheckout && modalActions}

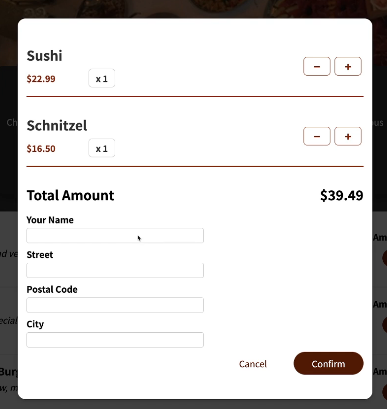
    </Modal>

  );

};

export default Cart;

Step 4 check result



220. Reading Form Values

Step 1: add Ref each input in Checkout.js

import { useRef, useState } from 'react';

import classes from './Checkout.module.css';

const isEmpty = (value) => value.trim() === '';

const isFiveChars = (value) => value.trim().length === 5;

const Checkout = (props) => {

  const [formInputsValidity, setFormInputsValidity] = useState({

    name: true,

    street: true,

    city: true,

    postalCode: true,

  });

  const nameInputRef = useRef();

  const streetInputRef = useRef();

  const postalCodeInputRef = useRef();

  const cityInputRef = useRef();

  const confirmHandler = (event) => {

    event.preventDefault();

    const enteredName = nameInputRef.current.value;

    const enteredStreet = streetInputRef.current.value;

    const enteredPostalCode = postalCodeInputRef.current.value;

    const enteredCity = cityInputRef.current.value;

    const enteredNameIsValid = !isEmpty(enteredName);

    const enteredStreetIsValid = !isEmpty(enteredStreet);

    const enteredCityIsValid = !isEmpty(enteredCity);

    const enteredPostalCodeIsValid = isFiveChars(enteredPostalCode);

    setFormInputsValidity({

      name: enteredNameIsValid,

      street: enteredStreetIsValid,

      city: enteredCityIsValid,

      postalCode: enteredPostalCodeIsValid,

    });

    const formIsValid =

      enteredNameIsValid &&

      enteredStreetIsValid &&

      enteredCityIsValid &&

      enteredPostalCodeIsValid;

    if (!formIsValid) {

      return;

    }

    // Submit cart data

  };

  const nameControlClasses = `${classes.control} ${

    formInputsValidity.name ? '' : classes.invalid

  }`;

  const streetControlClasses = `${classes.control} ${

    formInputsValidity.street ? '' : classes.invalid

  }`;

  const postalCodeControlClasses = `${classes.control} ${

    formInputsValidity.postalCode ? '' : classes.invalid

  }`;

  const cityControlClasses = `${classes.control} ${

    formInputsValidity.city ? '' : classes.invalid

  }`;

  return (

    <form className={classes.form} onSubmit={confirmHandler}>

      <div className={nameControlClasses}>

        <label htmlFor='name'>Your Name</label>

        <input type='text' id='name' ref={nameInputRef} />

        {!formInputsValidity.name && <p>Please enter a valid name!</p>}

      </div>

      <div className={streetControlClasses}>

        <label htmlFor='street'>Street</label>

        <input type='text' id='street' ref={streetInputRef} />

        {!formInputsValidity.street && <p>Please enter a valid street!</p>}

      </div>

      <div className={postalCodeControlClasses}>

        <label htmlFor='postal'>Postal Code</label>

        <input type='text' id='postal' ref={postalCodeInputRef} />

        {!formInputsValidity.postalCode && (

          <p>Please enter a valid postal code (5 characters long)!</p>

        )}

      </div>

      <div className={cityControlClasses}>

        <label htmlFor='city'>City</label>

        <input type='text' id='city' ref={cityInputRef} />

        {!formInputsValidity.city && <p>Please enter a valid city!</p>}

      </div>

      <div className={classes.actions}>

        <button type='button' onClick={props.onCancel}>

          Cancel

        </button>

        <button className={classes.submit}>Confirm</button>

      </div>

    </form>

  );

};

export default Checkout;

221 Adding Form Validation

Step 1: in checkout.js

import { useRef, useState } from 'react';

import classes from './Checkout.module.css';

const isEmpty = (value) => value.trim() === '';

const isFiveChars = (value) => value.trim().length === 5;

const Checkout = (props) => {

  const [formInputsValidity, setFormInputsValidity] = useState({

    name: true,

    street: true,

    city: true,

    postalCode: true,

  });

  const nameInputRef = useRef();

  const streetInputRef = useRef();

  const postalCodeInputRef = useRef();

  const cityInputRef = useRef();

  const confirmHandler = (event) => {

    event.preventDefault();

    const enteredName = nameInputRef.current.value;

    const enteredStreet = streetInputRef.current.value;

    const enteredPostalCode = postalCodeInputRef.current.value;

    const enteredCity = cityInputRef.current.value;

    const enteredNameIsValid = !isEmpty(enteredName);

    const enteredStreetIsValid = !isEmpty(enteredStreet);

    const enteredCityIsValid = !isEmpty(enteredCity);

    const enteredPostalCodeIsValid = isFiveChars(enteredPostalCode);

    setFormInputsValidity({

      name: enteredNameIsValid,

      street: enteredStreetIsValid,

      city: enteredCityIsValid,

      postalCode: enteredPostalCodeIsValid,

    });

    const formIsValid =

      enteredNameIsValid &&

      enteredStreetIsValid &&

      enteredCityIsValid &&

      enteredPostalCodeIsValid;

    if (!formIsValid) {

      return;

    }

    // Submit cart data

  };

  const nameControlClasses = `${classes.control} ${

    formInputsValidity.name ? '' : classes.invalid

  }`;

  const streetControlClasses = `${classes.control} ${

    formInputsValidity.street ? '' : classes.invalid

  }`;

  const postalCodeControlClasses = `${classes.control} ${

    formInputsValidity.postalCode ? '' : classes.invalid

  }`;

  const cityControlClasses = `${classes.control} ${

    formInputsValidity.city ? '' : classes.invalid

  }`;

  return (

    <form className={classes.form} onSubmit={confirmHandler}>

      <div className={nameControlClasses}>

        <label htmlFor='name'>Your Name</label>

        <input type='text' id='name' ref={nameInputRef} />

        {!formInputsValidity.name && <p>Please enter a valid name!</p>}

      </div>

      <div className={streetControlClasses}>

        <label htmlFor='street'>Street</label>

        <input type='text' id='street' ref={streetInputRef} />

        {!formInputsValidity.street && <p>Please enter a valid street!</p>}

      </div>

      <div className={postalCodeControlClasses}>

        <label htmlFor='postal'>Postal Code</label>

        <input type='text' id='postal' ref={postalCodeInputRef} />

        {!formInputsValidity.postalCode && (

          <p>Please enter a valid postal code (5 characters long)!</p>

        )}

      </div>

      <div className={cityControlClasses}>

        <label htmlFor='city'>City</label>

        <input type='text' id='city' ref={cityInputRef} />

        {!formInputsValidity.city && <p>Please enter a valid city!</p>}

      </div>

      <div className={classes.actions}>

        <button type='button' onClick={props.onCancel}>

          Cancel

        </button>

        <button className={classes.submit}>Confirm</button>

      </div>

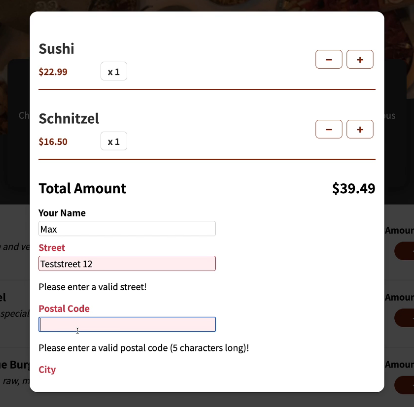
    </form>

  );

};

export default Checkout;

Step 2: Check result



222 Submitting & Sending Cart Data

Step 1: add SubmiOrderHandler in Cart.js

import { useContext, useState } from 'react';

import Modal from '../UI/Modal';

import CartItem from './CartItem';

import classes from './Cart.module.css';

import CartContext from '../../store/cart-context';

import Checkout from './Checkout';

const Cart = (props) => {

  const [isCheckout, setIsCheckout] = useState(false);

  const cartCtx = useContext(CartContext);

  const totalAmount = `$${cartCtx.totalAmount.toFixed(2)}`;

  const hasItems = cartCtx.items.length > 0;

  const cartItemRemoveHandler = (id) => {

    cartCtx.removeItem(id);

  };

  const cartItemAddHandler = (item) => {

    cartCtx.addItem(item);

  };

  const orderHandler = () => {

    setIsCheckout(true);

  };

  const submitOrderHandler = (userData) => {

    fetch('https://react-http-6b4a6.firebaseio.com/orders.json', {

      method: 'POST',

      body: JSON.stringify({

        user: userData,

        orderedItems: cartCtx.items

      })

    });

  };

  const cartItems = (

    <ul className={classes['cart-items']}>

      {cartCtx.items.map((item) => (

        <CartItem

          key={item.id}

          name={item.name}

          amount={item.amount}

          price={item.price}

          onRemove={cartItemRemoveHandler.bind(null, item.id)}

          onAdd={cartItemAddHandler.bind(null, item)}

        />

      ))}

    </ul>

  );

  const modalActions = (

    <div className={classes.actions}>

      <button className={classes['button--alt']} onClick={props.onClose}>

        Close

      </button>

      {hasItems && (

        <button className={classes.button} onClick={orderHandler}>

          Order

        </button>

      )}

    </div>

  );

  return (

    <Modal onClose={props.onClose}>

      {cartItems}

      <div className={classes.total}>

        <span>Total Amount</span>

        <span>{totalAmount}</span>

      </div>

      {isCheckout && (

        <Checkout onConfirm={submitOrderHandler} onCancel={props.onClose} />

      )}

      {!isCheckout && modalActions}

    </Modal>

  );

};

export default Cart;

Step 2: in Checkout.js (add props.onConfirm…)

import { useRef, useState } from 'react';

import classes from './Checkout.module.css';

const isEmpty = (value) => value.trim() === '';

const isFiveChars = (value) => value.trim().length === 5;

const Checkout = (props) => {

  const [formInputsValidity, setFormInputsValidity] = useState({

    name: true,

    street: true,

    city: true,

    postalCode: true,

  });

  const nameInputRef = useRef();

  const streetInputRef = useRef();

  const postalCodeInputRef = useRef();

  const cityInputRef = useRef();

  const confirmHandler = (event) => {

    event.preventDefault();

    const enteredName = nameInputRef.current.value;

    const enteredStreet = streetInputRef.current.value;

    const enteredPostalCode = postalCodeInputRef.current.value;

    const enteredCity = cityInputRef.current.value;

    const enteredNameIsValid = !isEmpty(enteredName);

    const enteredStreetIsValid = !isEmpty(enteredStreet);

    const enteredCityIsValid = !isEmpty(enteredCity);

    const enteredPostalCodeIsValid = isFiveChars(enteredPostalCode);

    setFormInputsValidity({

      name: enteredNameIsValid,

      street: enteredStreetIsValid,

      city: enteredCityIsValid,

      postalCode: enteredPostalCodeIsValid,

    });

    const formIsValid =

      enteredNameIsValid &&

      enteredStreetIsValid &&

      enteredCityIsValid &&

      enteredPostalCodeIsValid;

    if (!formIsValid) {

      return;

    }

    props.onConfirm({

      name: enteredName,

      street: enteredStreet,

      city: enteredCity,

      postalCode: enteredPostalCode,

    });

  };

  const nameControlClasses = `${classes.control} ${

    formInputsValidity.name ? '' : classes.invalid

  }`;

  const streetControlClasses = `${classes.control} ${

    formInputsValidity.street ? '' : classes.invalid

  }`;

  const postalCodeControlClasses = `${classes.control} ${

    formInputsValidity.postalCode ? '' : classes.invalid

  }`;

  const cityControlClasses = `${classes.control} ${

    formInputsValidity.city ? '' : classes.invalid

  }`;

  return (

    <form className={classes.form} onSubmit={confirmHandler}>

      <div className={nameControlClasses}>

        <label htmlFor='name'>Your Name</label>

        <input type='text' id='name' ref={nameInputRef} />

        {!formInputsValidity.name && <p>Please enter a valid name!</p>}

      </div>

      <div className={streetControlClasses}>

        <label htmlFor='street'>Street</label>

        <input type='text' id='street' ref={streetInputRef} />

        {!formInputsValidity.street && <p>Please enter a valid street!</p>}

      </div>

      <div className={postalCodeControlClasses}>

        <label htmlFor='postal'>Postal Code</label>

        <input type='text' id='postal' ref={postalCodeInputRef} />

        {!formInputsValidity.postalCode && (

          <p>Please enter a valid postal code (5 characters long)!</p>

        )}

      </div>

      <div className={cityControlClasses}>

        <label htmlFor='city'>City</label>

        <input type='text' id='city' ref={cityInputRef} />

        {!formInputsValidity.city && <p>Please enter a valid city!</p>}

      </div>

      <div className={classes.actions}>

        <button type='button' onClick={props.onCancel}>

          Cancel

        </button>

        <button className={classes.submit}>Confirm</button>

      </div>

    </form>

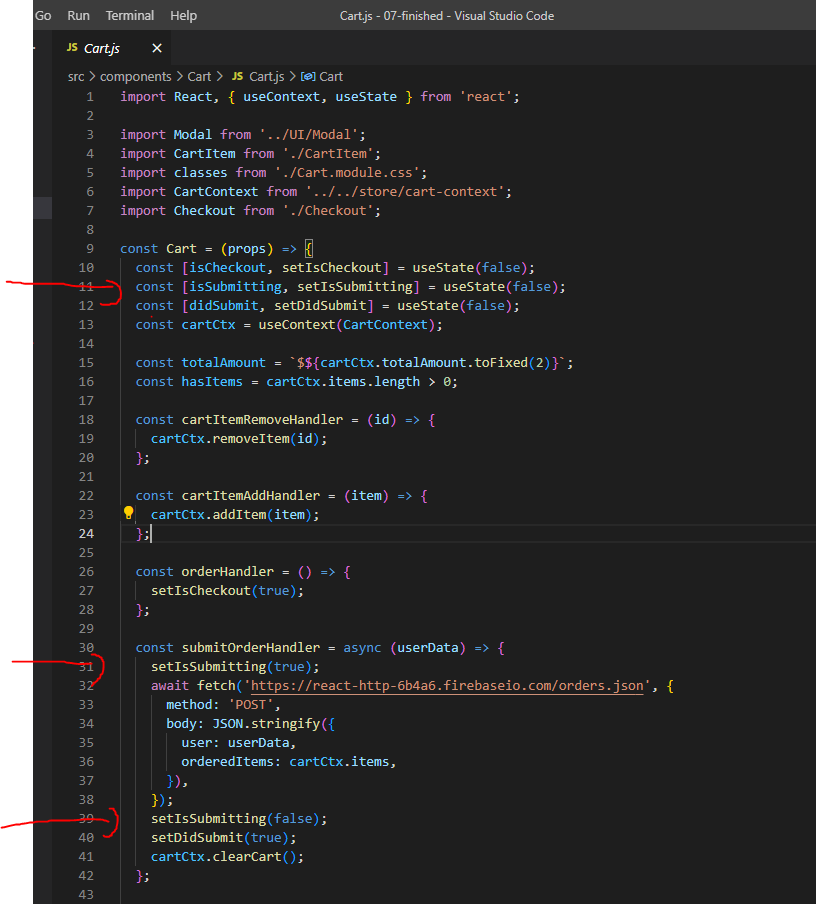
  );

};

export default Checkout;

223. Adding Better User Feedback

Step 1: in Cart.js (add setIsSubmitting)



import React, { useContext, useState } from 'react';

import Modal from '../UI/Modal';

import CartItem from './CartItem';

import classes from './Cart.module.css';

import CartContext from '../../store/cart-context';

import Checkout from './Checkout';

const Cart = (props) => {

  const [isCheckout, setIsCheckout] = useState(false);

  const [isSubmitting, setIsSubmitting] = useState(false);

  const [didSubmit, setDidSubmit] = useState(false);

  const cartCtx = useContext(CartContext);

  const totalAmount = `$${cartCtx.totalAmount.toFixed(2)}`;

  const hasItems = cartCtx.items.length > 0;

  const cartItemRemoveHandler = (id) => {

    cartCtx.removeItem(id);

  };

  const cartItemAddHandler = (item) => {

    cartCtx.addItem(item);

  };

  const orderHandler = () => {

    setIsCheckout(true);

  };

  const submitOrderHandler = async (userData) => {

    setIsSubmitting(true);

    await fetch('https://react-http-6b4a6.firebaseio.com/orders.json', {

      method: 'POST',

      body: JSON.stringify({

        user: userData,

        orderedItems: cartCtx.items,

      }),

    });

    setIsSubmitting(false);

    setDidSubmit(true);

    cartCtx.clearCart();

  };

  const cartItems = (

    <ul className={classes['cart-items']}>

      {cartCtx.items.map((item) => (

        <CartItem

          key={item.id}

          name={item.name}

          amount={item.amount}

          price={item.price}

          onRemove={cartItemRemoveHandler.bind(null, item.id)}

          onAdd={cartItemAddHandler.bind(null, item)}

        />

      ))}

    </ul>

  );

  const modalActions = (

    <div className={classes.actions}>

      <button className={classes['button--alt']} onClick={props.onClose}>

        Close

      </button>

      {hasItems && (

        <button className={classes.button} onClick={orderHandler}>

          Order

        </button>

      )}

    </div>

  );

  const cartModalContent = (

    <React.Fragment>

      {cartItems}

      <div className={classes.total}>

        <span>Total Amount</span>

        <span>{totalAmount}</span>

      </div>

      {isCheckout && (

        <Checkout onConfirm={submitOrderHandler} onCancel={props.onClose} />

      )}

      {!isCheckout && modalActions}

    </React.Fragment>

  );

  const isSubmittingModalContent = <p>Sending order data...</p>;

  const didSubmitModalContent = (

    <React.Fragment>

      <p>Successfully sent the order!</p>

      <div className={classes.actions}>

      <button className={classes.button} onClick={props.onClose}>

        Close

      </button>

    </div>

    </React.Fragment>

  );

  return (

    <Modal onClose={props.onClose}>

      {!isSubmitting && !didSubmit && cartModalContent}

      {isSubmitting && isSubmittingModalContent}

      {!isSubmitting && didSubmit && didSubmitModalContent}

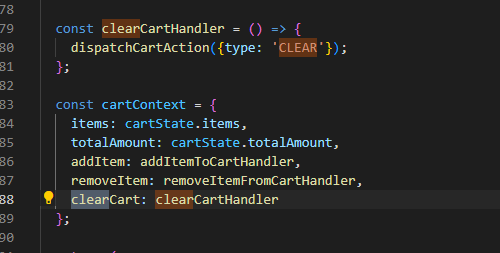
    </Modal>

  );

};

export default Cart;

Step 2: clear cart when order done (go to CartProvider.js)



import { useReducer } from 'react';

import CartContext from './cart-context';

const defaultCartState = {

  items: [],

  totalAmount: 0,

};

const cartReducer = (state, action) => {

  if (action.type === 'ADD') {

    const updatedTotalAmount =

      state.totalAmount + action.item.price \* action.item.amount;

    const existingCartItemIndex = state.items.findIndex(

      (item) => item.id === action.item.id

    );

    const existingCartItem = state.items[existingCartItemIndex];

    let updatedItems;

    if (existingCartItem) {

      const updatedItem = {

        ...existingCartItem,

        amount: existingCartItem.amount + action.item.amount,

      };

      updatedItems = [...state.items];

      updatedItems[existingCartItemIndex] = updatedItem;

    } else {

      updatedItems = state.items.concat(action.item);

    }

    return {

      items: updatedItems,

      totalAmount: updatedTotalAmount,

    };

  }

  if (action.type === 'REMOVE') {

    const existingCartItemIndex = state.items.findIndex(

      (item) => item.id === action.id

    );

    const existingItem = state.items[existingCartItemIndex];

    const updatedTotalAmount = state.totalAmount - existingItem.price;

    let updatedItems;

    if (existingItem.amount === 1) {

      updatedItems = state.items.filter(item => item.id !== action.id);

    } else {

      const updatedItem = { ...existingItem, amount: existingItem.amount - 1 };

      updatedItems = [...state.items];

      updatedItems[existingCartItemIndex] = updatedItem;

    }

    return {

      items: updatedItems,

      totalAmount: updatedTotalAmount

    };

  }

  if (action.type === 'CLEAR') {

    return defaultCartState;

  }

  return defaultCartState;

};

const CartProvider = (props) => {

  const [cartState, dispatchCartAction] = useReducer(

    cartReducer,

    defaultCartState

  );

  const addItemToCartHandler = (item) => {

    dispatchCartAction({ type: 'ADD', item: item });

  };

  const removeItemFromCartHandler = (id) => {

    dispatchCartAction({ type: 'REMOVE', id: id });

  };

  const clearCartHandler = () => {

    dispatchCartAction({type: 'CLEAR'});

  };

  const cartContext = {

    items: cartState.items,

    totalAmount: cartState.totalAmount,

    addItem: addItemToCartHandler,

    removeItem: removeItemFromCartHandler,

    clearCart: clearCartHandler

  };

  return (

    <CartContext.Provider value={cartContext}>

      {props.children}

    </CartContext.Provider>

  );

};

export default CartProvider;

225. Module Resources

You may want to **compare your code to mine** (e.g. to find + fix errors).

For that, you find **multiple code snapshots** for this module here in this Github repository: <https://github.com/academind/react-complete-guide-code/tree/17-practice-food-order-http-forms>