JSX Code:-

```jsx
import React, { useState, useEffect } from 'react';
import { Edit2, Plus, X, Check, Image, Package } from 'lucide-react';
import './new.css'; // Import the CSS file

const ProductCardGenerator = () => {
  const [products, setProducts] = useState([]);

  // Load products from localStorage on component mount
  useEffect(() => {
    const savedProducts = localStorage.getItem('products');
    if (savedProducts) {
      setProducts(JSON.parse(savedProducts));
    }
  }, []);

  // Save products to localStorage whenever products state changes
  useEffect(() => {
    localStorage.setItem('products', JSON.stringify(products));
  }, [products]);
  const [showForm, setShowForm] = useState(false);
  const [editingId, setEditingId] = useState(null);
  const [formData, setFormData] = useState({
    name: '',
    image: '',
    price: '',
    info: ''
  });

  const resetForm = () => {
    setFormData({ name: '', image: '', price: '', info: '' });
    setShowForm(false);
    setEditingId(null);
  };

  const handleSubmit = () => {
    if (!formData.name.trim() || !formData.price.trim()) {
      alert('Please fill in required fields (Name and Price)');
```

```
      return;
    }

    if (editingId) {
      setProducts(products.map(product =>
        product.id === editingId
          ? { ...formData, id: editingId, price:
parseFloat(formData.price) || 0 }
          : product
      ));
    } else {
      const newProduct = {
        id: Date.now(),
        ...formData,
        price: parseFloat(formData.price) || 0
      };
      setProducts([...products, newProduct]);
    }

    resetForm();
  };

  const handleEdit = (product) => {
    setFormData({
      name: product.name,
      image: product.image,
      price: product.price.toString(),
      info: product.info
    });
    setEditingId(product.id);
    setShowForm(true);
  };

  const handleDelete = (id) => {
    if (window.confirm('Are you sure you want to delete this product?')) {
      setProducts(products.filter(product => product.id !== id));
    }
  };

  const handleInputChange = (e) => {
```

```
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  return (
    <div className='body'>
      <h1 className='heading'>Product Card Generator</h1>
      <p className='sub-heading'>Create and manage beautiful product
cards</p>

      <button className='btn' onClick={() => setShowForm(true)}>
        <Plus size={20} /> Add New Product
      </button>

      {showForm && (
        <div className='modal' onClick={(e) => e.target ===
e.currentTarget && resetForm()}>
          <div className='modal-content'>
            <h2>{editingId ? 'Edit Product' : 'Add New Product'}</h2>
            <button className='close' onClick={resetForm}><X size={20}
/></button>

            <input
              type="text"
              name="name"
              value={formData.name}
              onChange={handleInputChange}
              placeholder="Product Name *"
              required
            />

            <input
              type="url"
              name="image"
              value={formData.image}
              onChange={handleInputChange}
              placeholder="Image URL (optional)"
            />

            <input
```

```
              type="number"
              name="price"
              value={formData.price}
              onChange={handleInputChange}
              placeholder="Price *"
              step="0.01"
              min="0"
              required
            />

            <textarea
              name="info"
              value={formData.info}
              onChange={handleInputChange}
              placeholder="Product Description (optional)"
              rows="3"
            />

            <div style={{display: 'flex', gap: '10px'}}>
              <button className='btn-reset'
onClick={resetForm}>Cancel</button>
              <button className='btn-submit' onClick={handleSubmit}>
                <Check size={18} /> {editingId ? 'Update Product' :
'Create Product'}
              </button>
            </div>
          </div>
        </div>
      )}

      {products.length === 0 ? (
        <div className='empty-state'>
          <Package size={80} />
          <h3>No products yet</h3>
          <p>Click "Add New Product" to create your first product card</p>
        </div>
      ) : (
        <div className='products-grid'>
          {products.map(product => (
            <div className='product-card' key={product.id}>
```

```
              <div className='card-image-container'>
                {product.image && (
                  <img
                    className='card-image'
                    src={product.image}
                    alt={product.name}
                    onError={(e) => {
                      e.target.style.display = 'none';
                      e.target.nextElementSibling.style.display = 'flex';
                    }}
                  />
                )}
                {!product.image && (
                  <div className={`image-placeholder ${!product.image ?
'visible' : ''}`}>
                    <Image size={48} />
                  </div>
                )}

                <div className='card-actions'>
                  <button className='action-btn edit-btn' onClick={() =>
handleEdit(product)} title="Edit Product">
                    <Edit2 size={16} />
                  </button>
                  <button className='action-btn delete-btn' onClick={() =>
handleDelete(product.id)} title="Delete Product">
                    <X size={16} />
                  </button>
                </div>
              </div>

              <div className='card-content'>
                <h3>{product.name}</h3>
                <div className='price-value'>
                  <p>${product.price.toFixed(2)}</p>
                </div>
                {product.info && <p className='info'>{product.info}</p>}
                <button>Add to Cart</button>
              </div>
            </div>
```

```
          ))}
        </div>
      )}
    </div>
  );
};

export default ProductCardGenerator;
```

Absolutely! Let me break down the `handleSubmit` function step by step with clear examples.

What handleSubmit Does

This function runs when someone clicks "Create Product" or "Update Product". It either adds a new product or updates an existing one.

Step-by-Step Breakdown

Step 1: Validation (Input Checking)

```
if (!formData.name.trim() || !formData.price.trim()) {
  alert('Please fill in required fields (Name and Price)');
  return;
}
```

What this does:

- `formData.name.trim()` - Removes spaces from the beginning and end of the name
- `!formData.name.trim()` - Checks if the name is empty after removing spaces
- `||` means "OR" - so if EITHER name OR price is empty, show an error
- `return` stops the function immediately if validation fails

Example:

```
// These would FAIL validation:
formData = { name: "   ", price: "" }     // Empty name (just spaces)
```

```
formData = { name: "iPhone", price: "" }   // Empty price
formData = { name: "", price: "100" }      // Empty name

// This would PASS validation:
formData = { name: "iPhone", price: "999" }
```

Step 2: Check if We're Editing or Adding

```
if (editingId) {
  // We're EDITING an existing product
} else {
  // We're ADDING a new product
}
```

How do we know?

- `editingId` is `null` when adding new products
- `editingId` contains a number when editing (the ID of the product being edited)

Step 3A: EDITING an Existing Product

```
if (editingId) {
  setProducts(products.map(product =>
    product.id === editingId
      ? { ...formData, id: editingId, price: parseFloat(formData.price) || 0 }
      : product
  ));
}
```

Let me break this down further:

The `map` function:

```
products.map(product => {
  // This runs for EACH product in the array
  if (product.id === editingId) {
    // This is the product we want to update
    return { ...formData, id: editingId, price: parseFloat(formData.price) || 0 }
  } else {
    // This is NOT the product we're editing, so keep it unchanged
```

```
    return product
  }
})
```

Example scenario:

```
// Current products array:
products = [
  { id: 1, name: "iPhone", price: 999, image: "", info: "Great phone" },
  { id: 2, name: "Samsung", price: 800, image: "", info: "Android phone" }
]

// We're editing product with id: 1
editingId = 1

// Form data from user:
formData = { name: "iPhone 15", price: "1099", image: "img.jpg", info: "Latest iPhone" }

// After map:
products = [
  { id: 1, name: "iPhone 15", price: 1099, image: "img.jpg", info: "Latest iPhone" }, //
UPDATED
  { id: 2, name: "Samsung", price: 800, image: "", info: "Android phone" }            //
UNCHANGED
]
```

Step 3B: ADDING a New Product

```
else {
  const newProduct = {
    id: Date.now(),
    ...formData,
    price: parseFloat(formData.price) || 0
  };
  setProducts([...products, newProduct]);
}
```

Breaking this down:

Creating the new product:

```
const newProduct = {
  id: Date.now(),                          // Unique ID using current timestamp
  ...formData,                             // Spread operator - copies all form data
  price: parseFloat(formData.price) || 0       // Convert price string to number
};
```

The spread operator (`...`):

```
// If formData is:
formData = { name: "iPad", price: "599", image: "ipad.jpg", info: "Tablet" }

// Then { ...formData } becomes:
{
  name: "iPad",
  price: "599",
  image: "ipad.jpg",
  info: "Tablet"
}
```

Example scenario:

```
// Current products:
products = [
  { id: 1, name: "iPhone", price: 999, image: "", info: "Great phone" }
]

// Form data:
formData = { name: "iPad", price: "599", image: "ipad.jpg", info: "Tablet" }

// New product created:
newProduct = {
  id: 1642584000000,  // Date.now() - current timestamp
  name: "iPad",
  price: 599,         // Converted to number
  image: "ipad.jpg",
  info: "Tablet"
}
```

```
// Final products array:
products = [
  { id: 1, name: "iPhone", price: 999, image: "", info: "Great phone" },
  { id: 1642584000000, name: "iPad", price: 599, image: "ipad.jpg", info: "Tablet" }
]
```

Step 4: Price Conversion

```
price: parseFloat(formData.price) || 0
```

What this does:

- `parseFloat()` converts text to a decimal number
- `|| 0` means "if conversion fails, use 0 instead"

Examples:

```
parseFloat("99.99") || 0    // Result: 99.99
parseFloat("abc") || 0      // Result: 0 (because "abc" can't be converted)
parseFloat("") || 0         // Result: 0 (empty string can't be converted)
parseFloat("50") || 0       // Result: 50
```

Step 5: Clean Up

```
resetForm();
```

This clears the form and closes it after successfully adding/updating.

Complete Flow Example

Adding a new product:

1. User fills form: `{ name: "MacBook", price: "1299", image: "mac.jpg", info: "Laptop" }`
2. Validation passes ✓
3. `editingId` is `null`, so we go to the "else" branch
4. Create new product with timestamp ID
5. Add to products array

6. Reset form

Editing existing product:

1. User clicks "Edit" on a product (this sets `editingId` and fills the form)
2. User changes name from "iPhone" to "iPhone 15"
3. Clicks "Update Product"
4. Validation passes ✓
5. `editingId` has a value, so we go to the "if" branch
6. Use `map` to find and update that specific product
7. Reset form

This function is the "brain" of the CRUD operations - it handles both Create and Update in one place!

Absolutely! Let me explain `handleEdit` and `handleDelete` in the same detailed way.

handleEdit Function

What handleEdit Does

This function runs when someone clicks the "Edit" button on a product card. It prepares the form to edit an existing product instead of creating a new one.

The Complete Function

```
const handleEdit = (product) => {
  setFormData({
    name: product.name,
    image: product.image,
    price: product.price.toString(),
    info: product.info
  });
  setEditingId(product.id);
  setShowForm(true);
};
```

Step-by-Step Breakdown

Step 1: Fill the Form with Existing Data

```
setFormData({
```

```
  name: product.name,
  image: product.image,
  price: product.price.toString(),
  info: product.info
});
```

What this does:

- Takes the product that was clicked on
- Copies all its data into the form
- `.toString()` converts the price number back to text (forms need text)

Example scenario:

```
// User clicks edit on this product:
const product = {
  id: 123,
  name: "iPhone 14",
  price: 999,
  image: "iphone.jpg",
  info: "Great smartphone"
}

// After setFormData runs:
formData = {
  name: "iPhone 14",        // Copied from product
  image: "iphone.jpg",      // Copied from product
  price: "999",             // Converted from number to string
  info: "Great smartphone"  // Copied from product
}
```

Step 2: Remember Which Product We're Editing

```
setEditingId(product.id);
```

What this does:

- Stores the ID of the product being edited
- This tells `handleSubmit` that we're editing, not adding new

Example:

```
// Before editing:
editingId = null  // null means "adding new product"

// After clicking edit:
editingId = 123   // Now we know we're editing product with ID 123
```

Step 3: Show the Form

```
setShowForm(true);
```

What this does:

- Opens the modal form so user can make changes
- The form will now show "Edit Product" instead of "Add New Product"

Complete Flow Example

User Journey:

1. User sees product card for "iPhone 14" priced at $999
2. User clicks the edit button (pencil icon)
3. `handleEdit` runs with the iPhone product data
4. Form opens with all the iPhone's current information filled in
5. Form title shows "Edit Product" instead of "Add New Product"
6. User can now modify any field and save changes

Before and After States:

```
// BEFORE handleEdit runs:
formData = { name: '', image: '', price: '', info: '' }  // Empty form
editingId = null                                  // Not editing anything
showForm = false                                  // Form is hidden

// AFTER handleEdit runs:
formData = { name: 'iPhone 14', image: 'iphone.jpg', price: '999', info: 'Great
smartphone' }
editingId = 123                                   // Editing product with ID 123
showForm = true                                   // Form is visible
```

---

handleDelete Function

What handleDelete Does

This function runs when someone clicks the "Delete" button (X icon) on a product card. It removes the product from the list after confirming with the user.

The Complete Function

```
const handleDelete = (id) => {
  if (window.confirm('Are you sure you want to delete this product?')) {
    setProducts(products.filter(product => product.id !== id));
  }
};
```

Step-by-Step Breakdown

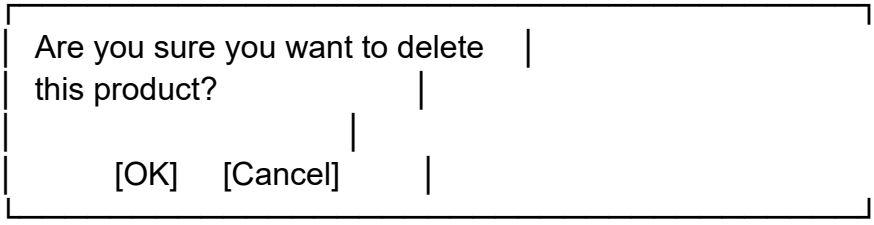
Step 1: Ask for Confirmation

```
if (window.confirm('Are you sure you want to delete this product?')) {
  // Only delete if user clicks "OK"
}
```

What this does:

- `window.confirm()` shows a browser popup with "OK" and "Cancel" buttons
- Returns `true` if user clicks "OK", `false` if user clicks "Cancel"
- This prevents accidental deletions

Visual example:



Step 2: Remove the Product (if confirmed)
setProducts(products.filter(product => product.id !== id));

Breaking down the `filter` method:

products.filter(product => product.id !== id)

What `filter` does:

- Goes through each product in the array
- Keeps products where the condition is `true`
- Removes products where the condition is `false`
- `product.id !== id` means "keep this product if its ID is NOT the one we want to delete"

Example scenario:

```
// Current products:
products = [
  { id: 1, name: "iPhone", price: 999 },
  { id: 2, name: "Samsung", price: 800 },  // ← User wants to delete this one
  { id: 3, name: "Google Pixel", price: 700 }
]

// User clicks delete on Samsung (id: 2)
// handleDelete(2) runs

// Filter process:
products.filter(product => product.id !== 2)

// Step by step:
// product.id = 1, is 1 !== 2? YES → Keep iPhone
// product.id = 2, is 2 !== 2? NO  → Remove Samsung
// product.id = 3, is 3 !== 2? YES → Keep Google Pixel

// Result:
products = [
  { id: 1, name: "iPhone", price: 999 },
  { id: 3, name: "Google Pixel", price: 700 }
```

```
]
```

Complete Flow Examples

Scenario 1: User Confirms Delete

```
// Initial state:
products = [
  { id: 1, name: "iPhone", price: 999 },
  { id: 2, name: "Samsung", price: 800 }
]

// User clicks delete on Samsung
handleDelete(2)

// 1. Confirmation popup appears
// 2. User clicks "OK"
// 3. window.confirm() returns true
// 4. Filter runs: keep products where id !== 2
// 5. Final state:
products = [
  { id: 1, name: "iPhone", price: 999 }
]
```

Scenario 2: User Cancels Delete

```
// Initial state:
products = [
  { id: 1, name: "iPhone", price: 999 },
  { id: 2, name: "Samsung", price: 800 }
]

// User clicks delete on Samsung
handleDelete(2)

// 1. Confirmation popup appears
// 2. User clicks "Cancel"
// 3. window.confirm() returns false
// 4. Code inside if-block doesn't run
// 5. Final state (unchanged):
products = [
```

```
  { id: 1, name: "iPhone", price: 999 },
  { id: 2, name: "Samsung", price: 800 }
]
```

How These Functions Work Together

1. **handleEdit** - Prepares existing data for modification
2. **handleSubmit** - Saves the modifications (knows it's editing because `editingId` is set)
3. **handleDelete** - Removes products completely

The cycle:

View Products → Click Edit → Modify in Form → Submit → Updated Product
    ↓
Click Delete → Confirm → Product Removed

These functions handle the "Update" and "Delete" parts of CRUD operations, making the app fully functional for managing products!

CSS Code:-

```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: sans-serif;
}

.body{
    display: flex;
    flex-direction: column;
    align-items: center;
    background: linear-gradient(60deg,blue,purple);
    min-height: 100vh;
    padding: 20px;
}
```

```css
.heading{
    color: white;
    font-size: 3.5rem;
    margin-top: 25px;
}

.sub-heading{
    color: rgb(69, 194, 44);
    font-size: 1.4rem;
}

.btn{
    font-size: 15px;
    background: linear-gradient(70deg,blue,violet);
    padding: 12px 25px;
    margin-top: 25px;
    cursor: pointer;
    border-radius: 20px;
    border: none;
    outline: none;
    color: white;
    transform: translateY(0px);
    box-shadow: 0 0 10px #1e90ff;
    transition: all 0.6s ease-in-out;
    display: flex;
    align-items: center;
    gap: 8px;
}

.btn:hover{
    box-shadow: 0 0 20px #1e90ff, 0 0 30px #1e90ff;
    transform: translateY(-10px);
}

.modal{
    z-index: 3;
    position: fixed;
    top: 0;
    left: 0;
    bottom: 0;
```

```
    right: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    background: rgba(77, 77, 77, 0.7);
    transition: all 0.4s;
}

.modal-content{
    display: flex;
    flex-direction: column;
    align-items: center;
    position: relative;
    background: #fff;
    width: 500px;
    max-width: 85%;
    min-height: 80vh;
    padding: 1em 2em;
    border-radius: 20px;
}

.modal-content h2{
    color: #333;
    font-size: 2.5em;
    border-bottom: 4px dotted blue;
}

.modal-content .close{
    position: absolute;
    top: 9px;
    right: 15px;
    background: transparent;
    border: none;
    border-radius: 50%;
    padding: 2px;
    transition: all 0.3s;
    cursor: pointer;
}

.modal-content .close:hover{
```

```
    background-color: blue;
    color: white;
}

.modal-content input{
  width: 100%;
  border: none;
  font-size: 16px;
  padding: 12px 12px 12px 0px;
  border-bottom: 2px solid #ccc;
  background: transparent;
  outline: none;
  margin-top: 10px;
  margin-bottom: 15px;
}

.modal-content textarea{
  width: 300px;
  height: 100px;
  outline: none;
  border: 3px solid #ccc;
  padding: 15px;
}

.modal-content textarea:focus {
  border-color: #999;
}

.btn-reset, .btn-submit {
    font-size: 15px;
    background: linear-gradient(70deg,blue,violet);
    padding: 12px 25px;
    margin: 10px;
    cursor: pointer;
    border-radius: 20px;
    border: none;
    outline: none;
    color: white;
    transform: translateY(0px);
    box-shadow: 0 0 10px #1e90ff;
```

```
    transition: all 0.6s ease-in-out;
    display: flex;
    align-items: center;
    gap: 8px;
}

.btn-reset:hover, .btn-submit:hover{
    box-shadow: 0 0 20px #1e90ff, 0 0 30px #1e90ff;
    transform: translateY(-10px);
}

.btn-reset {
    background: linear-gradient(70deg, #666, #999);
}

/* product card */

.products-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(320px, 1fr));
  gap: 2rem;
  margin-bottom: 3rem;
  width: 80%;
  max-height: 80vh;
  overflow-y: auto;
  margin-top: 30px;
}

/* Product Card */
.product-card {
  background: rgba(255, 255, 255, 0.1);
  border-radius: 24px;
  overflow: hidden;
  box-shadow: 0 8px 32px rgba(0, 0, 0, 0.3);
  transition: all 0.3s ease;
  position: relative;
  border: 1px solid rgba(255, 255, 255, 0.2);
  backdrop-filter: blur(10px);
}
```

```css
/* Add glow animation on hover */
.product-card:hover {
  animation: whiteBorderPulse 1.5s infinite alternate;
  transform: translateY(-8px);
}

@keyframes whiteBorderPulse {
  0% {
    border-color: rgba(255, 255, 255, 0.2);
    box-shadow: 0 0 0 rgba(255, 255, 255, 0.0);
  }
  100% {
    border-color: rgba(255, 255, 255, 1);
    box-shadow: 0 0 12px rgba(255, 255, 255, 0.6);
  }
}

.card-image-container {
  position: relative;
  height: 240px;
  overflow: hidden;
  background: linear-gradient(135deg, #f1f5f9 0%, #e2e8f0 100%);
}

.card-image {
  width: 100%;
  height: 100%;
  object-fit: cover;
  transition: all 0.3s ease;
}

.product-card:hover .card-image {
  transform: scale(1.1);
}

.image-placeholder {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
```

```
  height: 100%;
  display: none;
  align-items: center;
  justify-content: center;
  color: #666;
  background: linear-gradient(135deg, #f8fafc 0%, #f1f5f9 100%);
}

.image-placeholder.visible {
  display: flex;
}

.card-actions {
  position: absolute;
  top: 1rem;
  right: 1rem;
  display: flex;
  gap: 0.5rem;
  opacity: 0;
  transform: translateY(-10px);
  transition: all 0.3s ease;
}

.product-card:hover .card-actions {
  opacity: 1;
  transform: translateY(0);
}

.action-btn {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(10px);
  border: none;
  padding: 0.75rem;
  border-radius: 12px;
  cursor: pointer;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
  transition: all 0.3s ease;
  display: flex;
  align-items: center;
  justify-content: center;
```

```
}

.edit-btn:hover {
  background: #dbeafe;
  color: #2563eb;
  transform: scale(1.1);
}

.delete-btn:hover {
  background: #fef2f2;
  color: #dc2626;
  transform: scale(1.1);
}

.card-content{
  padding: 2rem;
}

.card-content h3{
  color: white;
  font-size: 1.5rem;
  text-align: center;
  text-transform: uppercase;
  margin-bottom: 0.3rem;
  font-weight: 700;
}

.card-content p{
  color: white;
  font-size: 1.6rem;
  margin-bottom: 5px;
}

.card-content .price-value{
  color: white;
  font-size: 1.6rem;
}

.card-content .price-value p{
  color: rgb(69, 194, 44);
```

```
  font-weight: bold;
}

.card-content .info{
  color: rgba(255, 255, 255, 0.8);
  font-size: 1rem;
  margin-bottom: 15px;
  text-align: center;
}

.card-content button{
  font-size: 15px;
  text-align: center;
  background: linear-gradient(70deg,blue,violet);
  padding: 8px 19px;
  margin-top: 25px;
  cursor: pointer;
  border-radius: 20px;
  border: none;
  outline: none;
  color: white;
  transform: translateY(0px);
  box-shadow: 0 0 10px #1e90ff;
  transition: all 0.6s ease-in-out;
  margin-left: auto;
  margin-right: auto;
  display: block;
}

.card-content button:hover{
  box-shadow: 0 0 20px #1e90ff, 0 0 30px #1e90ff;
  transform: translateY(-10px);
}

.empty-state {
  display: flex;
  flex-direction: column;
  align-items: center;
  color: white;
  text-align: center;
```

```
  margin-top: 4rem;
}

.empty-state h3 {
  font-size: 2rem;
  margin: 1rem 0;
}

.empty-state p {
  font-size: 1.2rem;
  opacity: 0.8;
}
```