## Implementing MVC pattern on Node. Js for our website Kontaniben

To implement the MVC pattern in your website using React for the frontend, Tailwind CSS for styling, and Node.js for the backend, the structure of the application is described below:

### Models:

Models represent the data of our application. In the backend (Node.js), these would typically be represented by database schemas or data structures.

In the context of our e-commerce website, models represent the data entities such as products, users, orders, etc. Here's an example of how we might implement a Product model in Node.js:

```
const Product = ({ id, name, price, description }) => {
  return {
    id,
    name,
    price,
    description
  };
};


module.exports = Product;
```

### Views:

Views represent the UI components of our application. In React, these would be React components responsible for rendering UI elements. In our React-based e-commerce website, views would be React components. Here's an example of a ProductCard component:

```
import React from 'react';


const ProductCard = ({ product }) => {
  return (
    <div className="border p-4">
      <h2 className="text-lg font-bold">{product.name}</h2>
      <p className="text-gray-500">${product.price}</p>
      <p>{product.description}</p>
      {/* Add buttons for actions like 'Add to Cart' */}
    </div>
  );
};


export default ProductCard;
```

**Controllers:**

Controllers act as intermediaries between models and views. In our case, they would handle user interactions and update models accordingly.

In our Node.js backend, controllers would handle routes and business logic. Here's an example of a controller for managing product-related operations:

```
const Product = require('../models/Product');

const products = [

  Product({ id: 1, name: 'Product 1', price: 19.99, description: 'Description of Product 1' }),

  Product({ id: 2, name: 'Product 2', price: 29.99, description: 'Description of Product 2' }),

  Product({ id: 3, name: 'Product 3', price: 39.99, description: 'Description of Product 3' })

];
```

```
// Controller methods
const getAllProducts = (req, res) => {
  res.json(products);
};

const getProductById = (req, res) => {
  const productId = parseInt(req.params.id);
  const product = products.find(p => p.id === productId);
  if (product) {
    res.json(product);
  } else {
    res.status(404).json({ message: 'Product not found' });
  }
};

module.exports = {
  getAllProducts,
  getProductById
};
```

**Integration:**

Now, we can integrate these components into our e-commerce website. For example, in a React component that displays a list of products:

```
import React, { useState, useEffect } from 'react';
import ProductCard from './ProductCard';

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

  useEffect(() => {
    // Fetch products from backend when component mounts
    fetch('/api/products')
      .then(response => response.json())
      .then(data => setProducts(data))
      .catch(error => console.error('Error fetching products:', error));
  }, []);

  return (
    <div className="grid grid-cols-3 gap-4">
      {products.map(product => (
        <ProductCard key={product.id} product={product} />
      ))}
    </div>
  );
};

export default ProductList;
```

In this setup, when a user interacts with the website (e.g., viewing product listings, adding items to the cart), the React components handle the presentation, and Node.js controllers handle the backend logic, ensuring that our application follows the MVC pattern.

By structuring our e-commerce website in this way, we can achieve a clean separation of concerns, making our codebase easier to maintain and scale as the project grows. Additionally, using React for the frontend, Tailwind CSS for styling, and Node.js for the backend provides a modern and efficient development stack.