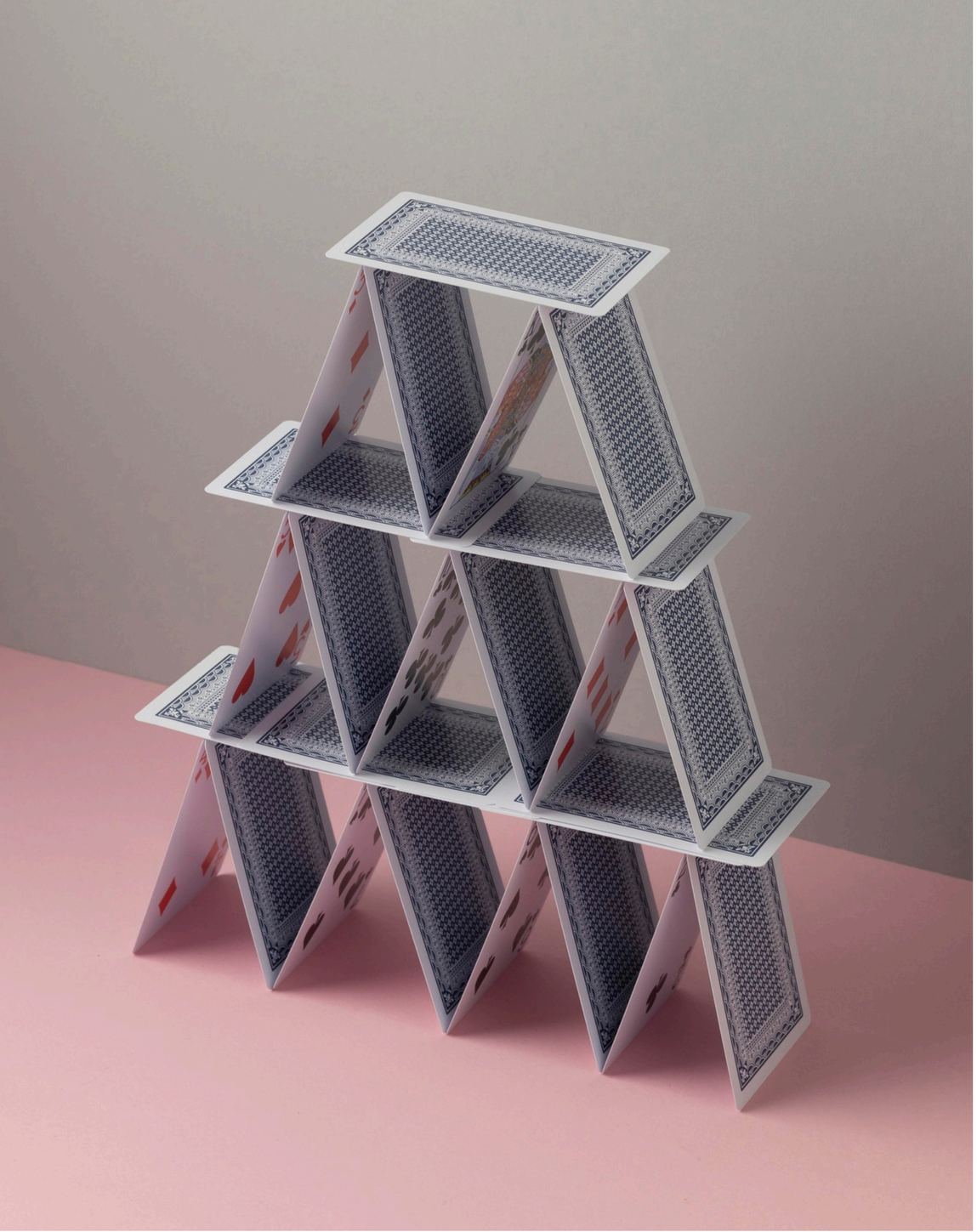
Shop EZ E-commerce Website Using the MERN Stack

TEAM MEMBERS

- SHOBAN BABU
- SHAIK IMRAN
- SHAIK KAIF
- SOWMYA

Introduction to Building Shop EZ

In this presentation, we will explore **Building Shop EZ**, a comprehensive guide to developing an **E-commerce website** using the **MERN Stack**. We will cover essential components, best practices, and key features to create a successful online store.



Understanding the MERN Stack

The **MERN Stack** consists of **MongoDB**, **Express.js**, **React.js**, and **Node.js**. Each technology plays a crucial role in building robust web applications, allowing for seamless integration and effective data management throughout the development process.

Architecture Diagram:

- Show the diagrams that explains the interaction between React.js (Frontend), Express.js (Backend), and MongoDB (Database).

Explain the flow:

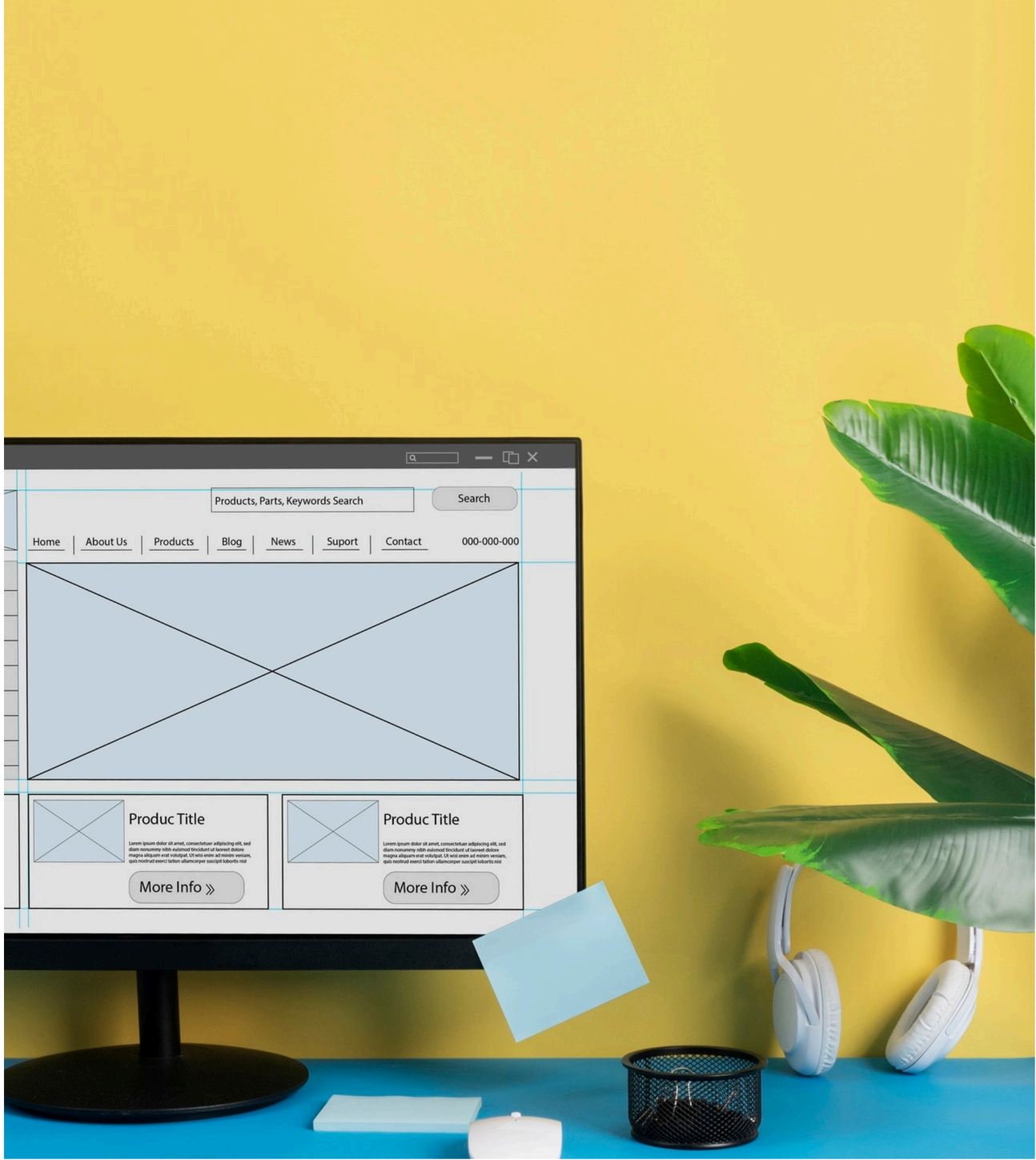
- User interacts with React frontend.
- React sends requests to the Node.js backend via Express.

Diagram showing the flow between:

- Front-end (React.js)
- Back-end (Express.js & Node.js)
- Database (MongoDB)
- Explanation of communication between components.
- Data is stored/retrieved from MongoDB.

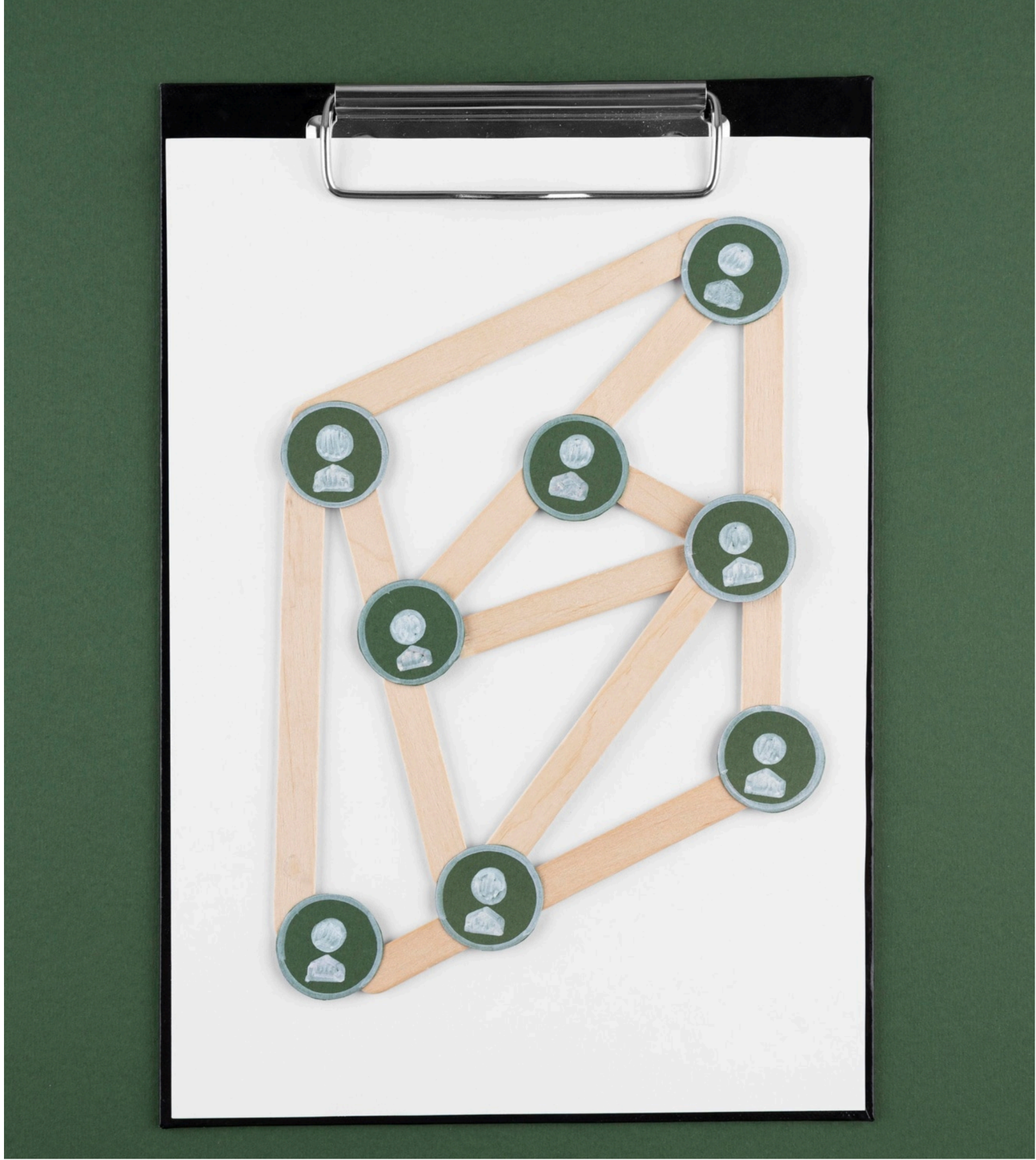
Setting Up Your Development Environment

To start building your e-commerce website, you need to set up your **development environment**. This includes installing **Node.js**, a code editor, and version control tools like **Git** to manage your project efficiently.



Designing the Database Schema

Designing a proper **database schema** is essential for organizing your data. Use **MongoDB** to create collections for products, users, and orders, ensuring that relationships between data are well-defined and easy to query.



Building the Backend with Node.js

The backend of your e-commerce site is built using **Node.js** and **Express.js**. This allows you to create RESTful APIs to handle requests, manage user authentication, and interact with the database effectively.

2. Backend Setup: (Node.js + Express)

Backend Structure:

1.backend/server.js: Entry point.

2.backend/models: Contains data models for MongoDB.

3.backend/routes: Routes for different API endpoints.

Backend Code for Product Model and API:

backend/models/Product.js

Define the product model using Mongoose:

backend/routes/productRoutes.js

Define API endpoints to interact with the product database:

backend/server.js

Creating the Frontend with React.js

With **React.js**, you can build a dynamic and responsive user interface. Utilize **components** to manage the layout and state of your application, ensuring a smooth user experience while browsing products.

Implementing User Authentication

User authentication is crucial for any e-commerce platform. Implement **JWT** (JSON Web Tokens) for secure user login and registration, ensuring that customer information is protected throughout their shopping experience.

User Authentication Flow:

```
const bcrypt = require('bcrypt');
const User = require('./models/User'); // Mongoose User model

app.post('/register', async (req, res) => {
  const hashedPassword = await bcrypt.hash(req.body.password, 10);
  const user = new User({
    username: req.body.username,
    password: hashedPassword
  });
  await user.save();
```

Managing Product Inventory

Effective product inventory management is key to a successful e-commerce site. Create functions to add, update, and delete products, ensuring that your inventory reflects accurate stock levels and product details.

React Product Listing and Product Details Page:

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import ProductList from './components/ProductList';
import ProductDetails from './components/ProductDetails';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<ProductList />} />
        <Route path="/product/:id" element={<ProductDetails />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Integrating Payment Gateway

To facilitate transactions, integrate a **payment gateway** such as **Stripe** or **PayPal**. This enables secure payment processing, allowing customers to complete their purchases easily and confidently.

Cart Functionality (React.js):

```
const Cart = () => {
  const [cart, setCart] = useState([]);

  const addToCart = (product) => {
    setCart([...cart, product]);
  };

  return (
    <div>
      <h2>Cart</h2>
      {cart.length === 0 ? <p>Your cart is empty</p> : <ul>{cart.map(item => <li
key={item.id}>{item.name}</li>)}</ul>}
    </div>
  );
};

export default Cart;
```

Testing Your E-commerce Site

Before launching, thoroughly **test** your e-commerce site. This includes unit testing, integration testing, and user acceptance testing to identify and fix any issues, ensuring a smooth operation for users.

Launching Your E-commerce Site

Once testing is complete, it’s time to **launch** your e-commerce website! Choose a reliable hosting provider, configure your domain, and deploy your application, making it accessible to your customers globally.

Conclusion and Next Steps

In conclusion, building an e-commerce website using the **MERN Stack** involves careful planning, development, and testing. Follow this guide to create a professional online store and consider exploring additional features to enhance user experience.

Thanks!