ChatGPT

Daraz-এর মত পূর্ণ-স্ট্যাক ই-কমার্স অ্যাপ গাইড (React, Redux Toolkit, Node.js, Express.js, MongoDB)

এই গাইডে আমরা একটি Daraz-এর মতো ই-কমার্স অ্যাপ তৈরি করার সম্পূর্ণ ধাপ-ধাপ প্রক্রিয়া দেখাব। এতে থাকবে React + Redux Toolkit দিয়ে ফ্রন্টএন্ড ও Node.js + Express + MongoDB দিয়ে ব্যাকএন্ড। গুরুত্বপূর্ণ ফিচারগুলো: **ইউজার অথেনটিকেশন (লগইন/লগআউট)**, **প্রোডাক্ট লিস্টিং**, **প্রোডাক্ট ডিটেইল পাতা**, **কার্ট ফাংশনালিটি**, **কার্ট ভিউ পেজ**, এবং **চেকআউট ও অর্ডার সামারি**। ব্যাকএন্ডে থাকবে **JWT-ভিত্তিক নিরাপদ অথেনটিকেশন**, **মডেল (User, Product, Order)**, **কার্ট ও অর্ডার এপিআই**, এবং প্রয়োজনীয় **মিডলওয়্যার (Auth, Error-Handling)**। আমরা **ফোল্ডার স্ট্রাকচার, রিয়েলিস্টিক API এন্ডপয়েন্ট, রেডাক্স স্টেট স্লাইস**, এবং **টোকেন সংরক্ষণ ও সেশন সুরক্ষা** বিষয়েও আলোচনা করব।

সমস্ত উদাহরণ কোডসহ এবং সাধারণ ব্যাখ্যাসহ সাজানো হবে, যাতে বাংলাদেশি ডেভেলপারদের জন্য প্রাসঙ্গিক হয় (যেমন টাকার ইউনিট ব্যবহার ইত্যাদি)।

১. প্রজেক্ট সেটআপ ও ফোল্ডার স্ট্রাকচার

প্রথমে প্রকল্পের ভিত্তি তৈরি করি। ডিফল্টভাবে আমরা ব্যাকএন্ড এবং ফ্রন্টএন্ড আলাদা দুটি ফোল্ডারে রাখব, যেমন:

```
project-root/
├── backend/
│   ├── config/        # পরিবেশগত ভেরিয়েবল ফাইল ইত্যাদি
│   ├── models/        # Mongoose মডেল
│   ├── routes/        # এক্সপ্রেস রাউটস
│   ├── controllers/   # রাউটারদের জন্য হ্যান্ডলার ফাংশন
│   ├── middleware/    # Auth ও Error হ্যান্ডলার
│   ├── db/            # ডাটাবেস কানেকশন (mongoose)
│   └── app.js         # প্রধান এক্সপ্রেস অ্যাপ
│
└── frontend/
    ├── public/
    ├── src/
    │   ├── components/   # UI কম্পোনেন্টস (Navbar, ProductCard, CartItem ইত্যাদি)
    │   ├── pages/        # পেজ কম্পোনেন্টস (Login, ProductList, CartPage ইত্যাদি)
    │   ├── redux/        # Redux Toolkit স্টোর ও স্লাইস
    │   ├── api/          # Axios ইন্সট্যান্স বা API কল ফাংশন
    │   ├── App.js        # রাউটার সংজ্ঞা
    │   └── index.js
    └── package.json
```

ব্যাকএন্ড ও ফ্রন্টএন্ড আলাদা রাখার সুবিধা হলো উন্নত স্কেলিবিলিটি ও পরিষ্কার পার্টিশন।

২. ব্যাকএন্ড (Node.js + Express + MongoDB)

ব্যাকএন্ডের জন্য প্রথমে একটি নতুন Node.js প্রজেক্ট তৈরি করুন। `backend` ফোল্ডারে যান এবং:

```
mkdir backend
cd backend
npm init -y
```

এখন প্রয়োজনীয় প্যাকেজ ইন্সটল করুন:

```
npm install express mongoose bcryptjs jsonwebtoken dotenv cors
npm install -D nodemon
```

- **Express**: Node.js এর জন্য লাইটওয়েট ওয়েব ফ্রেমওয়ার্ক [1]।
- **Mongoose**: MongoDB এর জন্য ODM লাইব্রেরি, যা সিমেন্টিকভাবে স্কীমা ও ভ্যালিডেশন দেয় [2]।
- **bcryptjs**: পাসওয়ার্ড হ্যাশিং-এর জন্য, সরাসরি টেক্সট পাসওয়ার্ড সঞ্চিত করলে নিরাপত্তা হুমকির সৃষ্টি হয়, তাই bcrypt দিয়ে হ্যাশ করে স্টোর করা হবে [3]।
- **jsonwebtoken**: JWT টোকেন জেনারেট এবং ভেরিফাই করার জন্য।
- **dotenv**: `.env` ফাইলে সংবেদনশীল কনফিগ রাখার জন্য। যেমন সিরেট, ডাটাবেস URL ইত্যাদি।
- **cors**: ফ্রন্টএন্ড ও ব্যাকএন্ড আলাদা হলে ক্রস-অরিজিন অনুরোধের জন্য প্রয়োজন।
- **nodemon** (dev-dependency): ডেভেলপমেন্টে কোড পরিবর্তনে স্বয়ংক্রিয় সার্ভার রিস্টার্ট [4]।

`package.json`-এ স্ক্রিপ্টে যুক্ত করতে পারেন:

```
"scripts": {
  "dev": "env-cmd -f ./config/dev.env nodemon src/app.js",
  "start": "node src/app.js"
}
```

এখানে `dev` মোডে `nodemon` প্রতিবার কোড পরিবর্তনে সার্ভার রিস্টার্ট করবে এবং পরিবেশ ভেরিয়েবল লোড করবে [5] [6]।

২.১ পরিবেশ ভেরিয়েবল (Config)

নতুন একটি `config` ফোল্ডার তৈরি করুন। এতে একটি `.env` ফাইল (যেমন `dev.env`) তৈরি করে লিখুন:

```
JWT_SECRET=ecommercewebapi
PORT=5000
MONGODB_URL=mongodb://127.0.0.1:27017/daraz_clone_db
```

উপরের `.env`-এ `JWT_SECRET` JWT সিক্রেট কী, `MONGODB_URL` আপনার লোকাল মঙ্গোডিবি ইউআরএল (বা ক্লাউড ইউআরএল), `PORT` সার্ভারের পোর্ট নির্দেশ করে। এই তথ্যগুলো সরাসরি কোডে না রেখে `.env`-এ রাখা নিরাপদ [6]।

Node-এ `dotenv` দিয়ে `.env` লোড করুন, উদাহরণ:

```
// src/app.js
require('dotenv').config({ path: './config/dev.env' });
const express = require('express');
```

২.২ ডাটাবেস কানেকশন

`db/mongoose.js` নামে ফাইল তৈরি করে মঙ্গোসেটআপ করুন:

```
// backend/db/mongoose.js
const mongoose = require('mongoose');
mongoose.connect(process.env.MONGODB_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
```

`app.js`-এ এই ফাইল রিকোয়্যার করুন যাতে সার্ভার চালুর আগে কানেকশন হয় [7]:

```
// backend/app.js
require('dotenv').config({ path: './config/dev.env' });
require('./db/mongoose');
const express = require('express');
const cors = require('cors');

const app = express();
app.use(express.json());
app.use(cors({ origin: 'http://localhost:3000', credentials: true })); // প্রয়োজনে ফ্রন্টএর ঠিকানা দিন
```

এখন অ্যাপ তৈরি হয়ে গেলো।

২.৩ মডেল (Models)

ইউজার মডেল (User)

`models/User.js` তৈরি করে স্কীমা বর্ণনা করুন:

```
// backend/models/User.js
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
```

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true, lowercase: true },
  password: { type: String, required: true },
}, { timestamps: true });

// পাসওয়ার্ড হ্যাশ করার জন্য pre-save হুক
userSchema.pre('save', async function (next) {
  if (this.isModified('password')) {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
  }
  next();
});

module.exports = mongoose.model('User', userSchema);
```

এখানে `bcrypt` দিয়ে পাসওয়ার্ড হ্যাশ করা হয়েছে, কারণ **সরাসরি টেক্সট পাসওয়ার্ড সংরক্ষণ করলে নিরাপত্তা হুমকি বাড়ে** [3]।

প্রোডাক্ট মডেল (Product)

`models/Product.js` ফাইল তৈরি করুন:

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

const productSchema = new mongoose.Schema({
  name:    { type: String, required: true },
  image:   { type: String, required: true },
  brand:   { type: String },
  category:{ type: String },
  price:   { type: Number, required: true },
  countInStock: { type: Number, default: 0 },
  description: { type: String },
}, { timestamps: true });

module.exports = mongoose.model('Product', productSchema);
```

এখানে প্রোডাক্টের নাম, ছবি, দাম, স্টক পরিমাণ, বিবরণ ইত্যাদি রাখা হয়েছে। প্রাথমিকভাবে কিছু নমুনা ডেটা দিয়ে শুরু করা যায় (seed data)।

অর্ডার মডেল (Order)

`models/Order.js` ফাইল:

```
// backend/models/Order.js
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  user:          { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  orderItems:   [
    {
      product:  { type: mongoose.Schema.Types.ObjectId, ref: 'Product',
required: true },
      name:     { type: String, required: true },
      qty:      { type: Number, required: true },
      price:    { type: Number, required: true },
      image:    { type: String },
    }
  ],
  shippingAddress: {
    address:   { type: String, required: true },
    city:      { type: String, required: true },
    postalCode:{ type: String, required: true },
    country:   { type: String, required: true },
  },
  paymentMethod: { type: String, default: 'Bkash' },
  itemsPrice:   { type: Number, required: true },
  taxPrice:     { type: Number, required: true },
  shippingPrice:{ type: Number, required: true },
  totalPrice:   { type: Number, required: true },
  isPaid:       { type: Boolean, default: false },
  paidAt:       { type: Date },
  createdAt:    { type: Date, default: Date.now },
}, { timestamps: true });

module.exports = mongoose.model('Order', orderSchema);
```

অর্ডারে কারও ইউজার রেফারেন্স, অর্ডারে থাকা আইটেমের তালিকা, শিপিং ঠিকানা ইত্যাদি থাকবে।

কার্ট মডেল (Cart)

ইচ্ছামতো `Cart` মডেল রাখতে পারেন, অথবা কার্ট ডেটা ইউজার মডেলে এড করতে পারেন। উদাহরণস্বরূপ আলাদা মডেল:

```
// backend/models/Cart.js
const mongoose = require('mongoose');

const cartSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true,
```

```
unique: true },
  cartItems: [
    {
      product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product',
required: true },
      qty:     { type: Number, required: true, default: 1 },
    }
  ]
}, { timestamps: true });

module.exports = mongoose.model('Cart', cartSchema);
```

এখানে প্রত্যেক ইউজারের জন্য একটি কার্ট থাকবে এবং সেটিতে প্রোডাক্টের আইডি ও পরিমাণ থাকবে।

২.৪ মিডলওয়্যার (Middleware)

অথেনটিকেশন মিডলওয়্যার

একটি `middleware/auth.js` ফাইল বানিয়ে JWT যাচাই কোড দিন:

```
// backend/middleware/auth.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const auth = async (req, res, next) => {
  let token;
  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select('-password');
      next();
    } catch (error) {
      res.status(401).json({ message: 'লগইন করা প্রয়োজন' });
    }
  } else {
    res.status(401).json({ message: 'অসঙ্গত টোকেন' });
  }
};

module.exports = auth;
```

এখানে আমরা `Authorization: Bearer <token>` হেডার চেক করে JWT ভেরিফাই করি এবং ভ্যালিড হলে `req.user`-এ ইউজার তথ্য সংরক্ষণ করি। নিরাপত্তার জন্য টোকেন ছাড়া বা ভ্যালিড না হলে 401 রেসপন্স দেব।

এ্যারর হ্যান্ডলিং মিডলওয়্যার

এক্সপ্রেসে ক্যাচ না হওয়া ত্রুটিগুলো ধরে রানটাইমে হ্যান্ডেল করতে একটি গ্লোবাল এ্যারর মিডলওয়্যার যুক্ত করুন:

```
// backend/middleware/errorHandler.js
const errorHandler = (err, req, res, next) => {
  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
  res.status(statusCode);
  res.json({ message: err.message, stack: process.env.NODE_ENV ===
'production' ? ' ' : err.stack });
};

module.exports = errorHandler;
```

এটি শেষের দিকে `app.use(errorHandler)` হিসেবে ব্যবহার হবে। এক্সপ্রেস নিজে সিংক্রোনাস এররগুলো `next(err)` ছাড়া ধরলেও, অ্যাসিঙ্ক্রোনাস কোডে ভুল করলে `next(err)` দিয়ে মেথডে পাস করতে হয় [8]। উপরের হ্যান্ডলার সব ত্রুটি পেয়ে JSON আকারে পাঠাবে।

৩. API রাউটস (Realistic Endpoints)

ব্যাকএন্ডে বাস্তবসম্মত এন্ডপয়েন্টগুলো ডিফাইন করব। আমাদের প্রয়োজনীয় রাউটস:

- **ইউজার রাউটস (User Routes)**
- `POST /api/users/register` – ইউজার রেজিস্টার (নাম, ইমেইল, পাসওয়ার্ড)
- `POST /api/users/login` – ইউজার লগইন (ইমেইল, পাসওয়ার্ড)
- `GET /api/users/profile` – লগইনকৃত ইউজারের প্রোফাইল (প্রোটেক্টেড)

- **প্রোডাক্ট রাউটস (Product Routes)**

- `GET /api/products` – সকল প্রোডাক্টের তালিকা (পাবলিক)
- `GET /api/products/:id` – একটি প্রোডাক্টের বিস্তারিত (পাবলিক)
- (ঐচ্ছিক) `POST/PUT/DELETE` প্রোডাক্ট – অ্যাডমিনের জন্য CRUD

- **কার্ট রাউটস (Cart Routes)**

- `GET /api/cart` – ইউজারের কার্ট ডেটা (প্রোটেক্টেড)
- `POST /api/cart` – কার্টে প্রোডাক্ট যোগ (প্রোটেক্টেড)
- `DELETE /api/cart/:productId` – প্রোডাক্ট রিমুভ (প্রোটেক্টেড)

- **অর্ডার রাউটস (Order Routes)**

- `POST /api/orders` – নতুন অর্ডার তৈরি (চেকআউট) (প্রোটেক্টেড)

- `GET /api/orders` – লগইন ইউজারের সব অর্ডার (প্রোটেক্টেড)
- `GET /api/orders/:id` – নির্দিষ্ট অর্ডারের বিস্তারিত (প্রোটেক্টেড)

উপরে বর্ণিত এন্ডপয়েন্টগুলো RESTful ডিজাইনের উদাহরণ। প্রতি রিসোর্স (ব্যবহারকারী, প্রোডাক্ট, কার্ট, অর্ডার) এর জন্য HTTP মেথডে প্রয়োজনীয় অ্যাকশন দেওয়া হয়। যেমন প্রোডাক্ট লিস্ট পেতে `GET /api/products` এবং যোগ করতে `POST /api/products` (অ্যাডমিনের জন্য) ইত্যাদি।

নিচে প্রতিটি রাউটের সারসংক্ষেপ দেয়া হলঃ

৩.১ ইউজার অথেনটিকেশন রাউটস

```
// backend/routes/userRoutes.js
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const User = require('../models/User');

// রেজিস্টার
router.post('/register', async (req, res, next) => {
  try {
    const { name, email, password } = req.body;
    const exists = await User.findOne({ email });
    if (exists) return res.status(400).json({ message: 'ইমেইল ইতিমধ্যেই নিবন্ধিত' });
    const user = await User.create({ name, email,
password }); // pre-save হুক দ্বারা হ্যাশ হবে
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: '1d' });
    res.status(201).json({
      _id: user._id, name: user.name, email: user.email, token
    });
  } catch (err) {
    next(err);
  }
});

// লগইন
router.post('/login', async (req, res, next) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (user && (await bcrypt.compare(password, user.password))) {
      const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: '1d' });
      res.json({
        _id: user._id, name: user.name, email: user.email, token
```

```
      });
    } else {
      res.status(401).json({ message: 'ইমেইল অথবা পাসওয়ার্ড ভুল' });
    }
  } catch (err) {
    next(err);
  }
});

// প্রোফাইল (প্রোটেক্টেড)
const auth = require('../middleware/auth');
router.get('/profile', auth, async (req, res) => {
  const user = req.user;
  res.json({ _id: user._id, name: user.name, email: user.email });
});

module.exports = router;
```

উপরের কোডে, রেজিস্টার এবং লগইন উভয়কেই `async/await` দিয়ে হ্যান্ডেল করা হয়েছে; ভুল হলে `next(err)` দিয়ে আমাদের **এ্যারর মিডলওয়্যার** প্রেরিত হবে [8] । সফল হলে JWT টোকেন জেনারেট করে রেসপন্সে পাঠানো হয়। এই টোকেন পরবর্তী প্রোটেক্টেড রাউটে হেডারে 'Bearer' হিসেবে ব্যবহার করতে হবে।

৩.২ প্রোডাক্ট এন্ডপয়েন্টস

```
// backend/routes/productRoutes.js
const express = require('express');
const router = express.Router();
const Product = require('../models/Product');

// সব প্রোডাক্ট
router.get('/', async (req, res, next) => {
  try {
    const products = await Product.find({});
    res.json(products);
  } catch (err) {
    next(err);
  }
});

// একটি প্রোডাক্ট ডিটেইল
router.get('/:id', async (req, res, next) => {
  try {
    const product = await Product.findById(req.params.id);
    if (product) res.json(product);
    else res.status(404).json({ message: 'প্রোডাক্ট পাওয়া যায়নি' });
  } catch (err) {
```

```
    next(err);
  }
});

module.exports = router;
```

প্রোডাক্ট লিস্ট পেতে `GET /api/products` এবং নির্দিষ্ট প্রোডাক্ট পেতে `GET /api/products/:id` ব্যবহার করা হবে। প্রয়োজনমত এডমিনের জন্য `POST`, `PUT`, `DELETE` এন্ডপয়েন্ট যোগ করা যেতে পারে।

৩.৩ কার্ট এন্ডপয়েন্টস

```
// backend/routes/cartRoutes.js
const express = require('express');
const router = express.Router();
const Cart = require('../models/Cart');
const Product = require('../models/Product');
const auth = require('../middleware/auth');

// ইউজারের কার্ট দেখুন / তৈরি করুন
router.get('/', auth, async (req, res, next) => {
  try {
    let cart = await Cart.findOne({ user:
req.user._id }).populate('cartItems.product');
    if (!cart) {
      cart = await Cart.create({ user: req.user._id, cartItems: [] });
    }
    res.json(cart);
  } catch (err) {
    next(err);
  }
});

// কার্টে প্রোডাক্ট যোগ করুন (বা পরিমাণ আপডেট)
router.post('/', auth, async (req, res, next) => {
  try {
    const { productId, qty } = req.body;
    let cart = await Cart.findOne({ user: req.user._id });
    if (!cart) {
      cart = new Cart({ user: req.user._id, cartItems: [] });
    }
    const itemIndex = cart.cartItems.findIndex(item => item.product.toString()
=== productId);
    if (itemIndex > -1) {
      // ইতিমধ্যে আইটেম আছে -> পরিমাণ আপডেট
      cart.cartItems[itemIndex].qty = qty;
    } else {
```

```
      // নতুন আইটেম যোগ
      cart.cartItems.push({ product: productId, qty });
    }
    await cart.save();
    res.json(cart);
  } catch (err) {
    next(err);
  }
});

// কার্ট থেকে প্রোডাক্ট রিমুভ করুন
router.delete('/:productId', auth, async (req, res, next) => {
  try {
    const cart = await Cart.findOne({ user: req.user._id });
    if (cart) {
      cart.cartItems = cart.cartItems.filter(item => item.product.toString() !
== req.params.productId);
      await cart.save();
      res.json(cart);
    } else {
      res.status(404).json({ message: 'কার্ট পাওয়া যায়নি' });
    }
  } catch (err) {
    next(err);
  }
});

module.exports = router;
```

এখানে ইউজার লগইন করা থাকলে তার কার্ট পাওয়া বা তৈরি করা হয়, এবং প্রোডাক্ট আইটেম যোগ/রিমুভ করা যায়। প্রতিবার হালনাগাদ কার্ট রেসপন্সে পাঠানো হয়।

৩.৪ অর্ডার এন্ডপয়েন্টস

```
// backend/routes/orderRoutes.js
const express = require('express');
const router = express.Router();
const Order = require('../models/Order');
const auth = require('../middleware/auth');

// নতুন অর্ডার (চেকআউট)
router.post('/', auth, async (req, res, next) => {
  try {
    const order = new Order({
      user: req.user._id,
      orderItems: req.body.orderItems,
```

```
      shippingAddress: req.body.shippingAddress,
      paymentMethod: req.body.paymentMethod || 'Bkash',
      itemsPrice: req.body.itemsPrice,
      taxPrice: req.body.taxPrice,
      shippingPrice: req.body.shippingPrice,
      totalPrice: req.body.totalPrice,
      isPaid: true,
      paidAt: Date.now(),
    });
    const createdOrder = await order.save();
    res.status(201).json(createdOrder);
  } catch (err) {
    next(err);
  }
});

// ইউজারের সব অর্ডার
router.get('/', auth, async (req, res, next) => {
  try {
    const orders = await Order.find({ user: req.user._id });
    res.json(orders);
  } catch (err) {
    next(err);
  }
});

// একটি অর্ডার ডিটেইল
router.get('/:id', auth, async (req, res, next) => {
  try {
    const order = await Order.findById(req.params.id);
    if (order) res.json(order);
    else res.status(404).json({ message: 'অর্ডার পাওয়া যায়নি' });
  } catch (err) {
    next(err);
  }
});

module.exports = router;
```

মিডলওয়্যার যুক্তকরণ: সব রাউটারগুলি `app.js`-এ মাউন্ট করুন এবং শেষে এরর হ্যান্ডলার:

```
// backend/app.js
const userRoutes = require('./routes/userRoutes');
const productRoutes = require('./routes/productRoutes');
const cartRoutes = require('./routes/cartRoutes');
const orderRoutes = require('./routes/orderRoutes');
```

```
const errorHandler = require('./middleware/errorHandler');

app.use('/api/users', userRoutes);
app.use('/api/products', productRoutes);
app.use('/api/cart', cartRoutes);
app.use('/api/orders', orderRoutes);
app.use(errorHandler);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

বিল্ড করা অ্যাপ্লিকেশন এখন রেডি। আপনি পোষ্টম্যান দিয়ে টেস্ট করতে পারেন।

৩.৫ নিরাপত্তা এবং সেরা অনুশীলন

- **পাসওয়ার্ড হ্যাশিং**: পাসওয়ার্ড `bcryptjs` দিয়ে হ্যাশ করুন [3]। ডাটাবেসে কখনই প্লেইন টেক্সটে রাখবেন না।
- **JWT সিক্রেট**: শক্তিশালী, ইউনিক সিক্রেট ব্যবহার করুন এবং `.env`-এ রাখুন [6]।
- **টোকেন সঞ্চয়**: ওয়েবের জন্য HTTP-only কুকি সবচেয়ে নিরাপদ কারণ এতে XSS আক্রমণের ঝুঁকি কম হয় [9]। তবে সহজের জন্য `localStorage`-এ রেখে থাকলে সতর্ক থাকতে হবে (XSS প্রতিরোধ)। আধুনিক ওয়েব অ্যাপে সাধারণত HttpOnly কুকি ব্যবহারই সুপারিশযোগ্য [10] [11]।
- **HTTPS**: প্রোডাকশনে সবসময় HTTPS ব্যবহার করুন [12]।
- **CORS ও CSRF**: কুকি ব্যবহার করলে CORS সঠিক কনফিগ করেন এবং SameSite ফ্ল্যাগ দিন। CSRF আক্রমণ রোধে সিকিউর ফ্ল্যাগসহ কুকি সেট করুন [9]।
- **এ্যারর প্রোটেকশন**: প্রডাকশনে স্ট্যাক ট্রেস লুকানোর ব্যবস্থা রাখুন (উপরের মিডলওয়্যার)। প্রয়োজনমত রেট লিমিটিং ও ইনপুট ভ্যালিডেশন যুক্ত করতে পারেন।
- **অথেনটিকেটেড এন্ডপয়েন্ট**: প্রোটেক্টেড রাউটে যথাযথ JWT ভেরিফিকেশন নিশ্চিত করুন। উদাহরণস্বরূপ, অ্যাসিঙ্ক্রোনাস ফাংশনে সমস্যা হলে `next(err)` ব্যবহার করে এক্সপ্রেস ভুল হ্যান্ডলারকে দিন [8]।

এই সিকিউরিটি গাইডলাইনগুলো অনুসরণ করলে অ্যাপটি তুলনামূলক সুরক্ষিত হবে।

৪. ফ্রন্টএন্ড (React + Redux Toolkit)

এবার React এবং Redux Toolkit দিয়ে ফ্রন্টএন্ড তৈরি করব।

৪.১ ক্লায়েন্ট সাইড সেটআপ

`frontend` ফোল্ডারে যান:

```
npx create-react-app frontend
cd frontend
npm install @reduxjs/toolkit react-redux react-router-dom axios
```

ডিরেক্টরি স্ট্রাকচার উদাহরণ:

```
frontend/src/
├── api/           # Axios ইনস্ট্যান্স বা API সেবার ফাইল
├── components/
│   ├── Navbar.js
│   ├── ProductCard.js
│   ├── CartItem.js
│   └── ...
├── pages/
│   ├── HomePage.js       # প্রোডাক্ট লিস্ট
│   ├── ProductPage.js    # প্রোডাক্ট ডিটেইল
│   ├── LoginPage.js
│   ├── CartPage.js
│   ├── CheckoutPage.js
│   └── ...
├── redux/
│   ├── store.js
│   ├── userSlice.js
│   ├── productSlice.js
│   ├── cartSlice.js
│   └── orderSlice.js
└── App.js
```

`src/store.js`-এ সমস্ত রিডিউসার যুক্ত করে স্টোর তৈরি করুন:

```
// frontend/src/redux/store.js
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './userSlice';
import productReducer from './productSlice';
import cartReducer from './cartSlice';
import orderReducer from './orderSlice';

const store = configureStore({
  reducer: {
    user: userReducer,
    products: productReducer,
    cart: cartReducer,
    order: orderReducer,
  },
});

export default store;
```

`configureStore` ব্যবহার করে সব স্লাইস এক করে স্টোর তৈরি করা হয় [13]। এরপর `index.js`-এ `Provider` দিয়ে র‍্যাপ করুন:

```
// frontend/src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import store from './redux/store';
import App from './App';

ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>,
  document.getElementById('root')
);
```

৪.২ রাউটিং (Routing)

`App.js`-এ পেজগুলো রাউট করুন:

```
// frontend/src/App.js
import { Routes, Route } from 'react-router-dom';
import HomePage from './pages/HomePage';
import ProductPage from './pages/ProductPage';
import CartPage from './pages/CartPage';
import CheckoutPage from './pages/CheckoutPage';
import LoginPage from './pages/LoginPage';

function App() {
  return (
    <>
      {/* Navbar কে সব পেজে দেখাতে পারেন */}
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/product/:id" element={<ProductPage />} />
        <Route path="/cart" element={<CartPage />} />
        <Route path="/checkout" element={<CheckoutPage />} />
        <Route path="/login" element={<LoginPage />} />
      </Routes>
    </>
  );
```

```
}

export default App;
```

৪.৩ রিডাক্স স্লাইস এবং লজিক

User Slice (লগইন/রেজিস্টার)

```
// frontend/src/redux/userSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// লগইন থাঙ্ক
export const loginUser = createAsyncThunk('user/loginUser', async ({ email,
password }, { rejectWithValue }) => {
  try {
    const res = await axios.post('/api/users/login', { email, password });
    return res.data;
  } catch (err) {
    return rejectWithValue(err.response.data.message);
  }
});

// রেজিস্টার থাঙ্ক
export const registerUser = createAsyncThunk('user/registerUser', async ({
name, email, password }, { rejectWithValue }) => {
  try {
    const res = await axios.post('/api/users/register', { name, email,
password });
    return res.data;
  } catch (err) {
    return rejectWithValue(err.response.data.message);
  }
});

const userSlice = createSlice({
  name: 'user',
  initialState: { userInfo: null, loading: false, error: null },
  reducers: {
    logout(state) {
      state.userInfo = null;
      localStorage.removeItem('userInfo');
    },
  },
  extraReducers: builder => {
    builder
```

```
      .addCase(loginUser.pending, (state) => { state.loading = true;
state.error = null; })
      .addCase(loginUser.fulfilled, (state, action) => {
        state.loading = false;
        state.userInfo = action.payload;
        localStorage.setItem('userInfo', JSON.stringify(action.payload));
      })
      .addCase(loginUser.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload;
      })
      // Register-সহ আরও কেস
      .addCase(registerUser.pending, (state) => { state.loading = true;
state.error = null; })
      .addCase(registerUser.fulfilled, (state, action) => {
        state.loading = false;
        state.userInfo = action.payload;
        localStorage.setItem('userInfo', JSON.stringify(action.payload));
      })
      .addCase(registerUser.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload;
      });
  },
});

export const { logout } = userSlice.actions;
export default userSlice.reducer;
```

স্লাইস তৈরিতে `createSlice` ব্যবহার করা হয় এবং এর `extraReducers`-এ `createAsyncThunk` এর পেন্ডিং, ফুলফিল্ড, রিজেক্টেড হ্যান্ডেল করা হয়। উপরে লগইন এবং রেজিস্টার উদাহরণ দেয়া হল। সফল হলে ইউজার ডেটা `localStorage`-এ সংরক্ষণ করা হয় যাতে রিফ্রেশ করলেও লগইন থাকে।

Redux Toolkit এর স্টোর কনফিগারেশন ও স্লাইস ব্যবহার সম্পর্কে আরও জানতে পারেন [14] [13]।

Product Slice

```
// frontend/src/redux/productSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// প্রোডাক্ট লিস্ট ফেচ
export const fetchProducts = createAsyncThunk('products/fetchAll', async () => {
  const res = await axios.get('/api/products');
  return res.data;
});
```

```
// প্রোডাক্ট ডিটেইল ফেচ
export const fetchProductById = createAsyncThunk('products/fetchById', async
(id) => {
  const res = await axios.get(`/api/products/${id}`);
  return res.data;
});

const productSlice = createSlice({
  name: 'products',
  initialState: { list: [], product: null, loading: false, error: null },
  reducers: {},
  extraReducers: builder => {
    builder
      .addCase(fetchProducts.pending, (state) => { state.loading = true;
state.error = null; })
      .addCase(fetchProducts.fulfilled, (state, action) => {
        state.loading = false;
        state.list = action.payload;
      })
      .addCase(fetchProducts.rejected, (state, action) => {
        state.loading = false;
        state.error = action.error.message;
      })
      .addCase(fetchProductById.pending, (state) => { state.loading = true;
state.error = null; })
      .addCase(fetchProductById.fulfilled, (state, action) => {
        state.loading = false;
        state.product = action.payload;
      })
      .addCase(fetchProductById.rejected, (state, action) => {
        state.loading = false;
        state.error = action.error.message;
      });
  },
});

export default productSlice.reducer;
```

এই স্লাইসে প্রোডাক্ট লিস্ট এবং একটি প্রোডাক্ট ডিটেইল করার থাঙ্ক দেওয়া হয়েছে।

Cart Slice

```
// frontend/src/redux/cartSlice.js
import { createSlice } from '@reduxjs/toolkit';
```

```
const initialState = {
  cartItems: JSON.parse(localStorage.getItem('cartItems')) || [],
};

const cartSlice = createSlice({
  name: 'cart',
  initialState,
  reducers: {
    addToCart(state, action) {
      const item = action.payload; // { productId, name, qty, price, image }
      const exist = state.cartItems.find(x => x.product === item.product);
      if (exist) {
        exist.qty = item.qty;
      } else {
        state.cartItems.push(item);
      }
      localStorage.setItem('cartItems', JSON.stringify(state.cartItems));
    },
    removeFromCart(state, action) {
      state.cartItems = state.cartItems.filter(x => x.product !==
action.payload);
      localStorage.setItem('cartItems', JSON.stringify(state.cartItems));
    },
    clearCart(state) {
      state.cartItems = [];
      localStorage.removeItem('cartItems');
    }
  }
});

export const { addToCart, removeFromCart, clearCart } = cartSlice.actions;
export default cartSlice.reducer;
```

কার্টে অ্যাড/রিমুভ করার জন্য এই স্লাইস ব্যবহার করা হচ্ছে। ইউজারের `localStorage`-এ কার্ট সিঙ্ক করে রাখা হয়েছে যাতে পেজ রিফ্রেশের পরও কার্ট থাকে।

Order Slice

```
// frontend/src/redux/orderSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

export const createOrder = createAsyncThunk('order/create', async (orderData, {
getState, rejectWithValue }) => {
  try {
    const { user: { userInfo } } = getState();
```

```
    const config = { headers: { Authorization: `Bearer ${userInfo.token}` } };
    const res = await axios.post('/api/orders', orderData, config);
    return res.data;
  } catch (err) {
    return rejectWithValue(err.response.data.message);
  }
});

const orderSlice = createSlice({
  name: 'order',
  initialState: { order: null, loading: false, error: null },
  reducers: {},
  extraReducers: builder => {
    builder
      .addCase(createOrder.pending, (state) => { state.loading = true;
state.error = null; })
      .addCase(createOrder.fulfilled, (state, action) => {
        state.loading = false;
        state.order = action.payload;
      })
      .addCase(createOrder.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload;
      });
  }
});

export default orderSlice.reducer;
```

`createOrder` থাঙ্ক চেকআউটের সময় ব্যাকএন্ডে অর্ডার পোষ্ট করে। এই এপিআই প্রোটেক্টেড, তাই `Authorization` হেডারে JWT পাঠাতে হয়।

৪.৪ কম্পোনেন্টস এবং পেজ টেমপ্লেট

লগইন পেজ (LoginPage.js)

```
// frontend/src/pages/LoginPage.js
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { loginUser } from '../redux/userSlice';
import { useNavigate } from 'react-router-dom';

function LoginPage() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const dispatch = useDispatch();
```

```
  const { userInfo, loading, error } = useSelector(state => state.user);
  const navigate = useNavigate();

  const submitHandler = async (e) => {
    e.preventDefault();
    await dispatch(loginUser({ email, password }));
  };

  if (userInfo) {
    navigate('/');
  }

  return (
    <div>
      <h2>লগইন করুন</h2>
      {error && <p style={{color:'red'}}>{error}</p>}
      <form onSubmit={submitHandler}>
        <input
          type="email" value={email} required
          placeholder="ইমেইল" onChange={(e)=> setEmail(e.target.value)} />
        <input
          type="password" value={password} required
          placeholder="পাসওয়ার্ড" onChange={(e)=> setPassword(e.target.value)} />
        <button type="submit">{loading ? 'লগইন...' : 'লগইন'}</button>
      </form>
    </div>
  );
}

export default LoginPage;
```

স্লাইসের `loginUser` থাঙ্ক ডিপ্যাচ করে বাকি সব কাজ হয়। `userInfo` ভ্যালিড হলে হোমে রিডিরেক্ট করানো হয়েছে।

প্রোডাক্ট লিস্ট পেজ (HomePage.js)

```
// frontend/src/pages/HomePage.js
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProducts } from '../redux/productSlice';
import ProductCard from '../components/ProductCard';

function HomePage() {
  const dispatch = useDispatch();
  const { list: products, loading, error } = useSelector(state =>
state.products);
```

```
  useEffect(() => {
    dispatch(fetchProducts());
  }, [dispatch]);

  return (
    <div>
      <h2>প্রোডাক্টসমূহ</h2>
      {loading ? <p>লোডিং...</p> : error ? <p>{error}</p> : (
        <div className="products-grid">
          {products.map(product => (
            <ProductCard key={product._id} product={product}/>
          ))}
        </div>
      )}
    </div>
  );
}

export default HomePage;
```

এই পেজে `fetchProducts` ডিস্প্যাচ করে পণ্যসমূহ লোড করা হয় এবং `ProductCard` কম্পোনেন্টে দেখানো হয়।

প্রোডাক্ট ডিটেইল পেজ (ProductPage.js)

```
// frontend/src/pages/ProductPage.js
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProductById } from '../redux/productSlice';
import { addToCart } from '../redux/cartSlice';

function ProductPage() {
  const { id } = useParams();
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { product, loading, error } = useSelector(state => state.products);
  const [qty, setQty] = useState(1);

  useEffect(() => {
    dispatch(fetchProductById(id));
  }, [dispatch, id]);

  const addToCartHandler = () => {
    dispatch(addToCart({
      product: product._id,
      name: product.name,
```

```
      image: product.image,
      price: product.price,
      qty,
    }));
    navigate('/cart');
  };

  if (loading) return <p>লোড হচ্ছে...</p>;
  if (error) return <p>{error}</p>;
  if (!product) return null;

  return (
    <div>
      <h2>{product.name}</h2>
      <img src={product.image} alt={product.name} width="200" />
      <p>মূল্য: ৳{product.price}</p>
      <p>{product.description}</p>
      <div>
        পরিমাণ:
        <select value={qty} onChange={(e) => setQty(Number(e.target.value))}>
          {[...Array(product.countInStock).keys()].map(x => (
            <option key={x+1} value={x+1}>{x+1}</option>
          ))}
        </select>
      </div>
      <button onClick={addToCartHandler}>কার্টে যোগ করুন</button>
    </div>
  );
}

export default ProductPage;
```

প্রোডাক্ট ডিটেইল লোড করে দেখানো হচ্ছে এবং **Add to Cart** বাটনে কার্ট স্লাইসে অ্যাড করা হচ্ছে।

কার্ট পেজ (CartPage.js)

```
// frontend/src/pages/CartPage.js
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { removeFromCart } from '../redux/cartSlice';
import { useNavigate } from 'react-router-dom';

function CartPage() {
  const { cartItems } = useSelector(state => state.cart);
  const dispatch = useDispatch();
  const navigate = useNavigate();
```

```
  const removeHandler = (id) => {
    dispatch(removeFromCart(id));
  };

  const checkoutHandler = () => {
    navigate('/checkout');
  };

  const totalPrice = cartItems.reduce((acc, item) => acc + item.price *
item.qty, 0);

  return (
    <div>
      <h2>আপনার কার্ট</h2>
      {cartItems.length === 0
        ? <p>কার্ট খালি আছে</p>
        : (
          <>
            {cartItems.map(item => (
              <div key={item.product}>
                <span>{item.name}</span>
                <span>৳{item.price} x {item.qty}</span>
                <button onClick={() => removeHandler(item.product)}>মুছে ফেলুন</
button>
              </div>
            ))}
            <h3>মোট মূল্য: ৳{totalPrice.toFixed(2)}</h3>
            <button onClick={checkoutHandler}>চেকআউট</button>
          </>
        )
      }
    </div>
  );
}

export default CartPage;
```

কার্ট আইটেম দেখানো হয়েছে, প্রতিটি আইটেম রিমুভ করা যাবে এবং টোটাল দেখানো হয়েছে। **Checkout** এ গিয়ে পেমেন্ট/অর্ডার সম্পন্ন করানো হবে।

চেকআউট পেজ (CheckoutPage.js)

```
// frontend/src/pages/CheckoutPage.js
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
```

```
import { createOrder } from '../redux/orderSlice';
import { clearCart } from '../redux/cartSlice';
import { useNavigate } from 'react-router-dom';

function CheckoutPage() {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { cartItems } = useSelector(state => state.cart);
  const { userInfo } = useSelector(state => state.user);

  const itemsPrice = cartItems.reduce((acc, item) => acc + item.price *
item.qty, 0);
  const shippingPrice = 60; // উদাহরণ স্বরূপ স্থির 60৳
  const taxPrice = itemsPrice * 0.15;
  const totalPrice = itemsPrice + shippingPrice + taxPrice;

  const placeOrder = async () => {
    const orderData = {
      orderItems: cartItems,
      shippingAddress: {
        address: 'বাংলাদেশ', city: 'ঢাকা', postalCode: '1230', country: 'Bangladesh'
      },
      paymentMethod: 'Bkash',
      itemsPrice,
      shippingPrice,
      taxPrice,
      totalPrice,
    };
    await dispatch(createOrder(orderData));
    dispatch(clearCart());
    navigate('/');
  };

  return (
    <div>
      <h2>অর্ডার সমারি</h2>
      <p>আইটেমস মুল্য: ৳{itemsPrice.toFixed(2)}</p>
      <p>শিপিং: ৳{shippingPrice.toFixed(2)}</p>
      <p>ট্যাক্স: ৳{taxPrice.toFixed(2)}</p>
      <h3>মোট: ৳{totalPrice.toFixed(2)}</h3>
      <button onClick={placeOrder}>অর্ডার প্লেস করুন</button>
    </div>
  );
}

export default CheckoutPage;
```

চূড়ান্ত অর্ডার সামারি দেখানো হচ্ছে এবং **Place Order** ক্লিক করলে ব্যাকএন্ডে অর্ডার ক্রিয়েট হয়। সফল হলে কার্ট ক্লিয়ার করে হোমে রিডিরেক্ট করা হয়েছে।

৪.৫ সিকিউরিটি ও টোকেন হ্যান্ডলিং

- **Axios কনফিগ**: যদি HTTP-only কুকি ব্যবহার করেন, তাহলে যেকোন `axios` অনুরোধে `{ withCredentials: true }` দিন। অন্যথায়, টোকেন হেডারে পাঠাতে পারেন:

```
// উদাহরণ: axios ইন্সট্যান্স তৈরি
import axios from 'axios';
const axiosInstance = axios.create({
  baseURL: '/api',
  withCredentials: true,
});
export default axiosInstance;
```

অথবা টোকেন হেডারে যোগ করুন:

```
axios.defaults.headers.common['Authorization'] = `Bearer ${userInfo.token}
`;
```

- **লোকাল স্টোরেজ**: স্থানীয় স্টোরেজে `userInfo` রাখলে, XSS আক্রমণের ঝুঁকি থাকে [15]। নিরাপত্তার জন্য HttpOnly কুকি ব্যবহার করতে পারেন [9]। তবে সিম্পল টিউটোরিয়ালে লোকালস্টোরেজ ব্যবহার করা হলেও, নেটিভ পাসওয়ার্ড ইত্যাদি কখনই সেভ করবেন না।
- **HTTPS**: প্রোডাকশনে অবশ্যই HTTPS দিয়ে হোস্ট করুন।
- **সেশন হার্ডেনিং**: টোকেনের মেয়াদ সীমিত করুন, প্রয়োজনে রিফ্রেশ টোকেন সিস্টেম ব্যবহার করুন। টোকেন ম্যানেজমেন্টের জন্য উভয় দিকের নিয়ন্ত্রণ রাখুন [16]।

৫. পরিশেষে

উপরের প্রতিটি ধাপে কোড ও ব্যাখ্যা দেওয়া হয়েছে যাতে একটি সম্পূর্ণ Daraz-এর মত MERN-স্ট্যাক ই-কমার্স অ্যাপ তৈরি করা যায়। ব্যাকএন্ডে মডেল, রাউট, মিডলওয়্যার ও সিকিউরিটি সেবা করা হয়েছে, এবং ফ্রন্টএন্ডে React + Redux Toolkit দিয়ে পেজ ও স্টেট ম্যানেজমেন্ট দেখানো হয়েছে।

গুরুত্বপূর্ণ টিপস

- পরিবেশ ফাইল `.env` এ সিক্রেট ও ডাটাবেস ইউআরএল রাখুন [6]।
- পাসওয়ার্ড বেঁধে হ্যাশ করুন [3]।
- JWT HttpOnly কুকি দিয়ে সেন্ড করুন, নিরাপত্তা বাড়াতে পারেন [9]।
- স্টেট ম্যানেজমেন্টে Redux Toolkit ব্যবহার করুন, স্লাইস ও `createAsyncThunk` ব্যবহার উপকারী [14] [13]।
- কার্ট ও ইউজার ডেটা `localStorage` বা Redux Persist দিয়ে পার্সিস্ট রাখতে পারেন, তবে টোকেন/পাসওয়ার্ড এড়িয়ে চলুন।
- বাস্তব ব্যবহারে **মূল্য প্রদর্শন** এ বাংলাদেশের টাকার ইউনিট (“৳”) ব্যবহার করা যেতে পারে।

এই গাইডে দেখানো স্টেপগুলো মেনে চললেই আপনার Laravel/MERN-স্ট্যাক ভিত্তিক ই-কমার্স ওয়েব অ্যাপের প্রাথমিক সব ফিচার তৈরি হয়ে যাবে। এরপর সহজেই পেমেন্ট গেটওয়ে, অনুমোদিত অ্যাডমিন প্যানেল, মাল্টি-স্টোর লজিক ইত্যাদি যোগ করতে পারবেন। শুভ ডেভেলপমেন্ট!

উৎসঃ MERN ই-কমার্স টিউটোরিয়াল ও JWT সিকিউরিটি গাইড থেকে তথ্য সংগ্রহ করা হয়েছে [17] [9] [13] [8]।

---

[1] [2] [3] [4] [5] [6] [7] [17] How I built an E-commerce API with NodeJs, Express and MongoDB(Part 1) | by Bassit Owolabi | Geek Culture | Medium
https://medium.com/geekculture/how-i-built-an-e-commerce-api-with-nodejs-express-and-mongodb-7b42b5253ffb

[8] Express error handling
https://expressjs.com/en/guide/error-handling.html

[9] [10] [11] [12] [15] [16] Comparing JWT Authentication Strategies: HTTP-Only Cookies vs LocalStorage | by Nijat Aliyev | Jun, 2025 | Medium
https://medium.com/@developer.nijat/comparing-jwt-authentication-strategies-http-only-cookies-vs-localstorage-05254ed99722

[13] [14] Understanding Redux Toolkit: A Practical Guide with an E-commerce Application | by Vikas Ipar | Medium
https://medium.com/@vikasipar/understanding-redux-toolkit-a-practical-guide-with-an-e-commerce-application-938cf07d38a0