3. E-commerce Website

Creating an E-commerce Website using the MEAN stack (MongoDB, Express.js, Angular, Node.js) with product listings, shopping cart, and payment processing is a substantial project. Here's a high-level overview of how to get started, and I'll provide some code snippets to guide you.

Project Setup and Structure

Set up a new project folder and structure for your E-commerce Website. Install the required Node.js packages and create a basic Angular application.

 # Create a new Angular application

ng new ecommerce-website

- Backend (Node.js & Express.js)

Create the backend of your E-commerce Website using Node.js and Express.js.

Installation of Packages

Install the necessary packages for Express.js, Mongoose, Passport.js for user authentication, and more.

npm install express mongoose passport passport-local bcryptjs jsonwebtoken

Setting up Express.js

Create your Express.js server, set up middleware, and handle routes.

- **javascript**

```
// server.js

const express = require('express');
const mongoose = require('mongoose');
const passport = require('passport');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();

// Middleware
app.use(bodyParser.json());
app.use(cors());

// Database connection
mongoose.connect('mongodb://localhost/ecommerce', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
  useCreateIndex: true,
});

// Passport.js for authentication
app.use(passport.initialize());
require('./config/passport')(passport);

// Routes
app.use('/api/users', require('./routes/users'));
app.use('/api/products', require('./routes/products'));
app.use('/api/orders', require('./routes/orders'));

const port = process.env.PORT || 3000;

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Define Schemas and Models

Create Mongoose schemas and models for products, user profiles, and orders. These models will define how data is structured and stored in MongoDB.

- **javascript**

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

const productSchema = new mongoose.Schema({
  name: String,
  description: String,
  price: Number,
  // Add more fields as needed
});

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

// Create similar models for User and Order
```

Routes

Define routes for CRUD operations for products, user profiles, and orders. For example, creating a route for product listing:

- **javascript**

```
// routes/products.js
const express = require('express');
```

```
const router = express.Router();

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


// Get a list of products

router.get('/', (req, res) => {

  Product.find()

    .then((products) => res.json(products))

    .catch((err) => console.log(err));

});


// Create similar routes for other operations (create, update, delete)
```

- ## Frontend (Angular)

Create the frontend of your E-commerce Website using Angular. Design the user interface, implement product listings, shopping cart, and user authentication.

Design and UI

Design the user interface for your E-commerce Website using Angular components, templates, and styles.

Implement Product Listings

Create components to display product listings and details. Fetch product data from the backend API.

- **typescript**

```
// product-list.component.ts
import { Component, OnInit } from '@angular/core';
import { ProductService } from './product.service';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
})
export class ProductListComponent implements OnInit {
  products: any;

  constructor(private productService: ProductService) {}

  ngOnInit() {
    this.productService.getProducts().subscribe((data) => {
      this.products = data;
    });
```

```
  }
}
```

Shopping Cart

Implement a shopping cart where users can add products, view their cart, and proceed to checkout.

User Authentication

Set up user authentication using Angular components and Passport.js on the server side. Create registration and login forms.

- **typescript**

```
// auth.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  constructor(private http: HttpClient) {}
```

```
  register(user) {
    return this.http.post('/api/users/register', user);
  }

  login(user) {
    return this.http.post('/api/users/login', user);
  }
}
```

MongoDB

Create a MongoDB database to handle product listings, user profiles, and order data.

Payment Processing

Implement payment processing using a payment gateway or service. This typically involves integrating with a third-party API like Stripe or PayPal to handle payment transactions.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle user authentication, cart management, and product listings properly. Implement payment processing and security measures for handling payments.

Building a full-fledged E-commerce Website is a substantial task, and you should consider other aspects like order management, inventory control, and scalability as you progress with your project.