



دانشگاه صنعتی نوشیروانی بابل

دانشکده برق و کامپیوتر

گزارش پژوهه کارشناسی رشته کامپیوتر گرایش نرم افزار

وب اپلیکیشن فروشگاه لباس

دانشجو

ماهان جعفری پالندی

۹۷۳۱۱۲۰۳۸

استاد پژوه

آقای دکتر ولی نتاج

فهرست مطالب

3.....	مقدمه
4.....	۱. فصل اول - پیاده سازی بخش فرانت اند
4.....	۱-۱. ایجاد پروژه
4.....	۱-۲. کامپوننت Footer و Header
5.....	۱-۳. لیست محصولات.....
6.....	۱-۴. پیاده سازی Router
7.....	۱-۵. کامپوننت امتیازدهی.....
7.....	۱-۶. صفحه هر محصول.....
9.....	۲. فصل دوم - ارائه و دریافت داده
9.....	۲-۱. معماری پروژه.....
10.....	۲-۲. سرور Express
11.....	۲-۳. اسکریپت های پروژه
11.....	۲-۴. درخواست داده از سمت فرانت اند
13.....	۳. فصل سوم - پایگاه داده
13.....	۳-۱. شروع کار با پایگاه داده
13.....	۳-۲. ارتباط با پایگاه داده از طریق برنامه
14.....	۳-۳. مدل کردن داده ها
16.....	۳-۴. دریافت محصولات
16.....	۳-۵. بهبود پروژه
17.....	۴. فصل چهارم - مدیریت State ها
17.....	۴-۱. مفهوم اولیه و آشنایی با Redux
17.....	۴-۲ . تعریف Redux
18.....	۴-۳ . محصولات Slice
19.....	۴-۴ . کامپوننت Loading و Message
21.....	۵. فصل پنجم - سبد خرید
21.....	۵-۱. شروع کار با سبد خرید
21.....	۵-۲. تابع اضافه شدن به سبد خرید
22.....	۵-۳. نمایش تعداد آیتم سبد خرید
22.....	۵-۴. صفحه سبد خرید

23.....	۵-۵. حذف از سبد خرید
24.....	۶. فصل ششم - ویژگی های اضافه
24.....	۶-۱. پیاده سازی بخش بازخورد.....
26.....	۶-۲. پیاده سازی بخش صفحه بندي.....
27.....	۶-۳. پیاده سازی اسلайдر.....
28.....	۶-۴. داده های Meta صفحات
29.....	۶-۵. پیاده سازی بخش جستجو.....
31.....	منابع

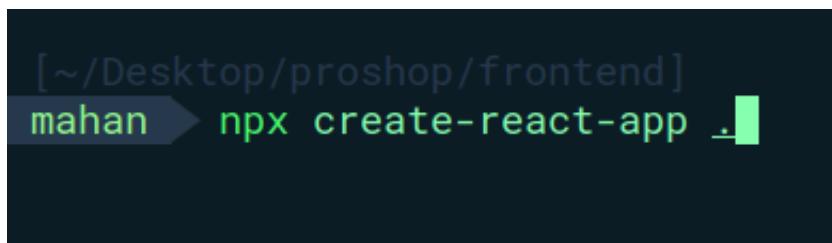
مقدمه

پروژه حاضر یک وب اپلیکیشن فروشگاه لباس است. در این اپلیکیشن شما می‌توانید کارهایی از قبیل ثبت سفارش، جستجوی لباس و خرید لباس انجام دهید. روند کار بدین صورت است که شما محصولات اپلیکیشن را می‌توانید در صفحه اصلی سایت مشاهده کنید و وارد صفحه محصول شوید و محصول را به سبد خرید اضافه کنید. اینجا نیاز است شما حتماً احراز هویت انجام داده باشید. سپس شما می‌توانید هزینه محصول را پرداخت کنید و محصول را خریداری کنید. این اپلیکیشن دارای ویژگی‌های بیشتری نظیر ثبت نظرات، صفحه بندی و غیره نیز می‌باشد که می‌توانید در فصل‌های پیش رو مطالعه می‌کنید. برای دسترسی به سورس کد پروژه هم می‌توانید به این [لينك](#) مراجعه کنید.

۱. فصل اول - پیاده سازی بخش فرانت‌اند

۱-۱. ایجاد پروژه

برای پیاده سازی بخش فرانت‌اند از کتابخانه React استفاده شده است. روند کار به این صورت است که ما یک پوشه به نام proshop که نام فروشگاه است ایجاد می‌کنیم. حال نیاز داریم درون این پوشه یک پوشه به نام frontend ایجاد کنیم که تمامی فایل‌های مربوط به بخش فرانت‌اند سایت درون این پوشه قرار بگیرد. برای ایجاد یک پروژه با React باید از پکیج create-react-app استفاده کنیم که خود با زبان Javascript پیاده سازی شده است. برای این منظور نیاز است شما حتما Node که یک runtime برای زبان Javascript است را بر روی سیستم خود نصب داشته باشید. حال با زدن دستوری که در تصویر شماره ۱ مشاهده می‌کنید می‌توانید پروژه خود را ایجاد کنید. حال که که پروژه ما با موفقیت ایجاد شد می‌توانیم پیاده سازی بخش فرانت‌اند را شروع کنیم.



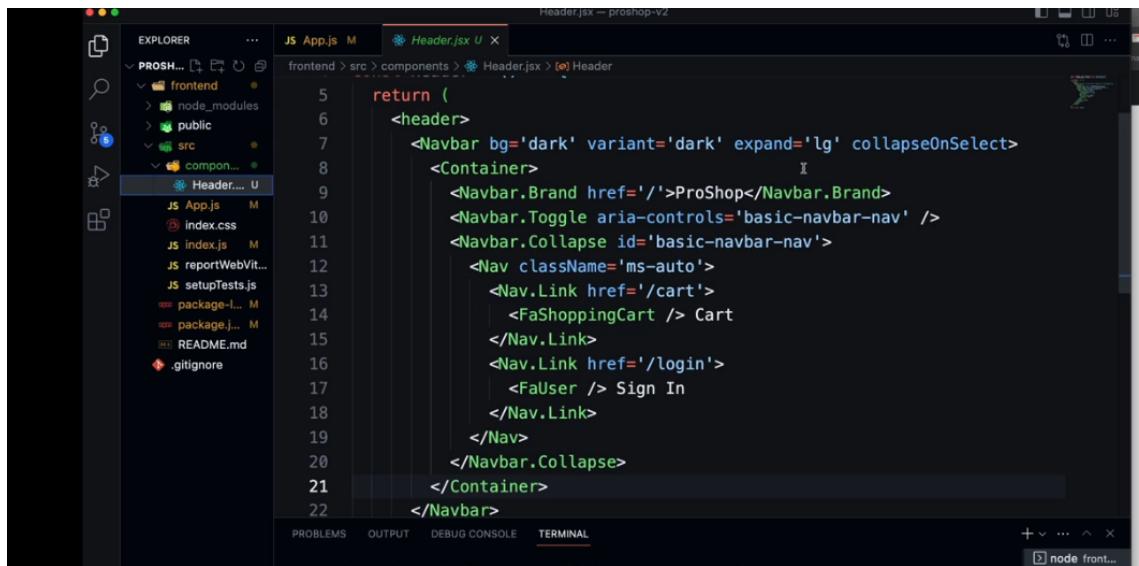
```
[~/Desktop/proshop/frontend]
mahan ➔ npx create-react-app .
```

تصویر ۱ - ایجاد پروژه

۲-۱. کامپوننت Header و Footer

برای پیاده سازی کامپوننت‌های مختلف سایت از کتابخانه Bootstrap که یک کتابخانه برای پیاده سازی رابط کاربری است استفاده می‌کنیم. در پوشه پروژه پوشه‌ای به نام components ایجاد می‌کنیم که تمامی کامپوننت‌های بخش ظاهر سایت درون آن قرار می‌گیرد. داخل این

پوشه دو کامپوننت به نام های Footer و Header به صورتی که در تصاویر زیر نشان داده شده است پیاده سازی می کنیم.



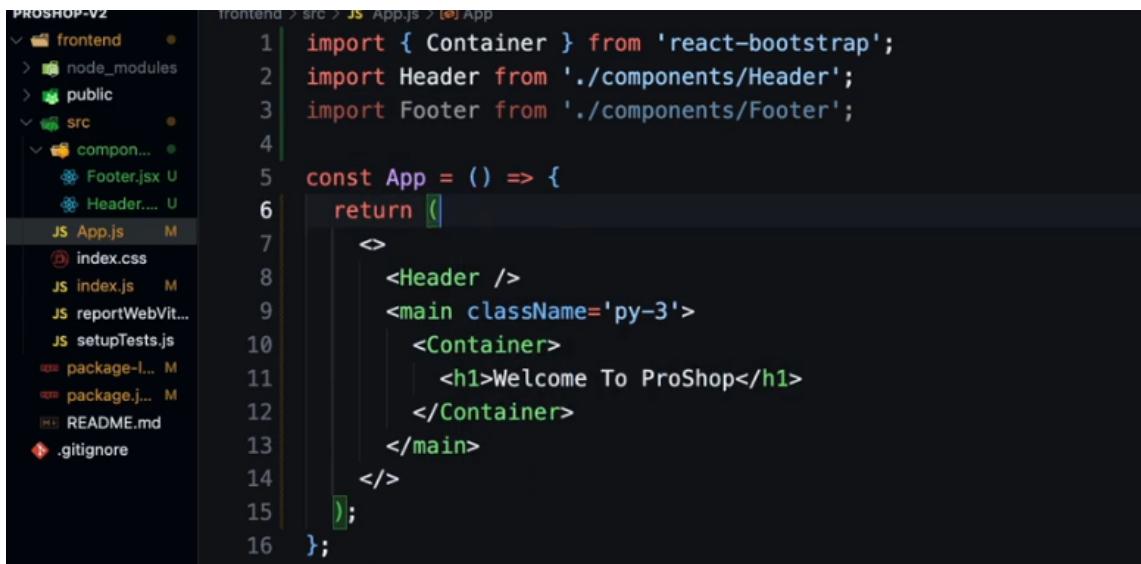
The screenshot shows the VS Code interface with the Header.js file open in the editor. The code defines a functional component that returns a JSX structure. It includes a Navbar with a dark variant, a Container, and a Nav bar item with a href to '/cart'. The code uses several Bootstrap and React components like Navbar, Container, Nav, and Nav.Link.

```

Header.js - proshop-v2
frontend > src > components > Header.js > Header
5   return (
6     <header>
7       <Navbar bg='dark' variant='dark' expand='lg' collapseOnSelect>
8         <Container>
9           <Navbar.Brand href='/'>ProShop</Navbar.Brand>
10          <Navbar.Toggle aria-controls='basic-navbar-nav' />
11          <Navbar.Collapse id='basic-navbar-nav'>
12            <Nav className='ms-auto'>
13              <Nav.Link href='/cart'>
14                <FaShoppingCart /> Cart
15              </Nav.Link>
16              <Nav.Link href='/login'>
17                <FaUser /> Sign In
18              </Nav.Link>
19            </Nav>
20          </Navbar.Collapse>
21        </Container>
22      </Navbar>

```

تصویر ۲ - کامپوننت Header



The screenshot shows the VS Code interface with the App.js file open in the editor. The code defines a functional component named App that returns a JSX structure. It includes a Header component, a main container with a py-3 class, and a h1 heading with the text "Welcome To ProShop". The code uses the Container component from react-bootstrap and imports Header and Footer components from the components directory.

```

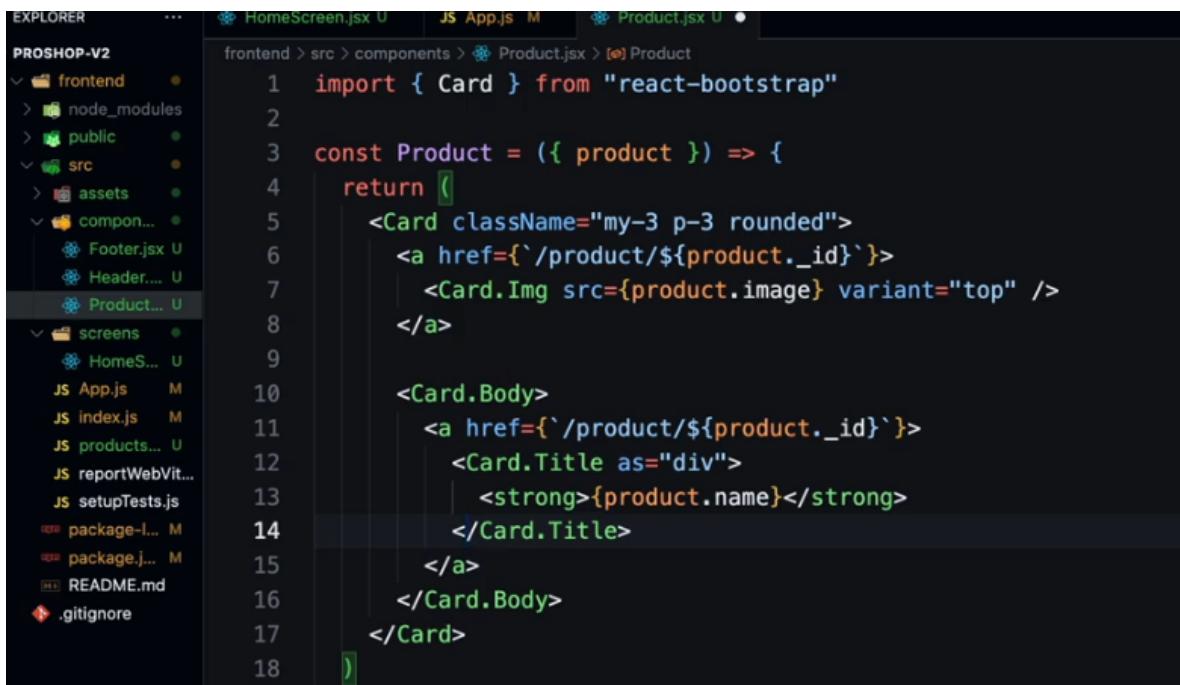
PROSHOP-V2
frontend > src > JS App.js > App
1 import { Container } from 'react-bootstrap';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4
5 const App = () => {
6   return (
7     <>
8       <Header />
9       <main className='py-3'>
10         <Container>
11           <h1>Welcome To ProShop</h1>
12         </Container>
13       </main>
14     </>
15   );
16 };

```

تصویر ۳ - کامپوننت Footer

۱-۳. لیست محصولات

حال که پیاده سازی دو کامپوننت قبلی به اتمام رسید نیاز است بتوانیم محصولات اصلی که لباس ها هستند را روی صفحه اصلی سایت پیاده سازی کنیم. برای این کار نیاز است یک پوشه به نام screens ایجاد کنیم که تمامی صفحات اصلی سایت درون این پوشه قرار می‌گیرند. حال یک کامپوننت به نام HomeScreen ایجاد می‌کنیم و به ترتیب در آن کامپوننت های Product و Header و Footer را ایمپورت می‌کنیم. اینجا یک کامپوننت به نام Product ایجاد می‌کنیم و فعلاً که پایگاه داده ای را تعریف نکرده ایم از یک سری اطلاعات پیش‌فرض برای نمایش محصولات استفاده می‌کنیم.



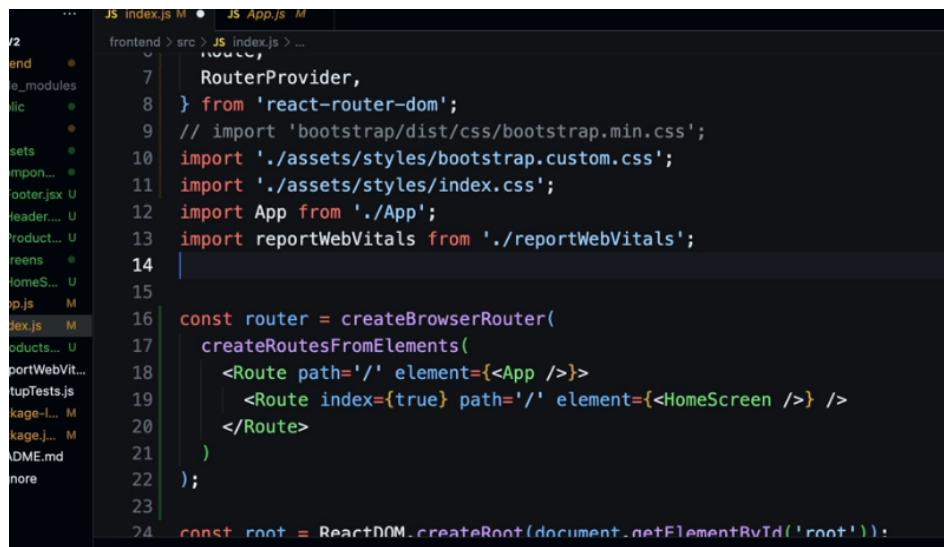
The screenshot shows a code editor with an 'EXPLORER' sidebar on the left containing project files like 'frontend', 'node_modules', 'public', 'src', 'assets', 'components', 'screens', and various JS files. The main area displays the 'Product.js' file:

```
frontend > src > components > Product.js > Product
1 import { Card } from "react-bootstrap"
2
3 const Product = ({ product }) => {
4   return (
5     <Card className="my-3 p-3 rounded">
6       <a href={`/product/${product._id}`}>
7         <Card.Img src={product.image} variant="top" />
8       </a>
9
10      <Card.Body>
11        <a href={`/product/${product._id}`}>
12          <Card.Title as="div">
13            <strong>{product.name}</strong>
14          </Card.Title>
15        </a>
16      </Card.Body>
17    </Card>
18  )
```

تصویر ۴ - کامپوننت Product

۴- پیاده سازی Router

برای این پروژه چون چندین صفحه داریم و میخواهیم بین آنها جابجا شویم نیاز است Router به پروژه خود اضافه کنیم. اینکار را به وسیله کتابخانه react-router-dom انجام می‌دهیم. Router را به پروژه اضافه می‌کنیم و تنها صفحه‌ای که تا الان پیاده سازی کردیم یعنی صفحه Home را به آن اضافه می‌کنیم.



```
JS index.js M JS App.js M
frontend > src > JS index.js > ...
  7   RouterProvider,
  8 } from 'react-router-dom';
  9 // import 'bootstrap/dist/css/bootstrap.min.css';
10 import './assets/styles/bootstrap.custom.css';
11 import './assets/styles/index.css';
12 import App from './App';
13 import reportWebVitals from './reportWebVitals';
14
15
16 const router = createBrowserRouter(
17   createRoutesFromElements(
18     <Route path='/' element={<App />}>
19       <Route index={true} path='/' element={<HomeScreen />} />
20     </Route>
21   )
22 );
23
24 const root = ReactDOM.createRoot(document.getElementById('root'));

```

تصویر ۵ - بخش Router

۵- کامپوننت امتیازدهی

در این بخش کامپوننت Rating را پیاده سازی می‌کنیم به صورتی که کاربر بتواند ۱ تا ۵ ستاره به هر محصولی که بازدید می‌کند بدهد. در اینجا نیاز به آیکون ستاره برای پیاده سازی این کامپوننت داریم برای همین پکیج react-icons را به پروژه اضافه کنیم.

```
<FaRegStar /> }
    </span>
    <span>
        { value >= 2 ? <FaStar /> : value >= 1.5 ? <FaStarHalfAlt /> :
        <FaRegStar /> }
    </span>
    <span>
        { value >= 3 ? <FaStar /> : value >= 2.5 ? <FaStarHalfAlt /> :
        <FaRegStar /> }
    </span>
    <span>
        { value >= 4 ? <FaStar /> : value >= 1.5 ? <FaStarHalfAlt /> :
        <FaRegStar /> }
    </span>
    <span>
        { value >= 2 ? <FaStar /> : value >= 1.5 ? <FaStarHalfAlt /> :
        <FaRegStar /> }
    </span>
```

تصویر۴ - کامپونت Rating

۱-۶. صفحه هر محصول

حال نیاز داریم یک صفحه به پروژه اضافه کنیم که اطلاعات هر محصول به صورت جزئی در آن صفحه قرار گرفته باشد و کاربر بتواند قیمت محصول را مشاهده کند و اگر خواست آن را به سبد خرید خود اضافه کند. برای پیادهسازی این صفحه ابتدا باید صفحه ProductScreen را به پروژه اضافه کنیم و سپس کامپونت ProductScreen را پیاده سازی کنیم.

```
src > JS index.js > [ej] router
import App from './App';
import reportWebVitals from './reportWebVitals';
import HomeScreen from './screens/HomeScreen';
import ProductScreen from './screens/ProductScreen';

const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path='/' element={<App />}>
      <Route index={true} path='/' element={<HomeScreen />}>
        <Route path='/product/:id' element={<ProductScreen />}>
      </Route>
    )
);

```

تصویر ۷ - اضافه کردن Router

```
    />
  </ListGroup.Item>
  <ListGroup.Item>Price: ${product.price}</ListGroup.Item>
</ListGroup>
</Col>
<Col md={3}>
  <Card>
    <ListGroup variant='flush'>
      <ListGroup.Item>
        <Row>
          <Col>Price:</Col>
          <Col>
            <strong>${product.price}</strong>
          </Col>
        </Row>
      </ListGroup.Item>
    </ListGroup>
  </Card>
</Col>

```

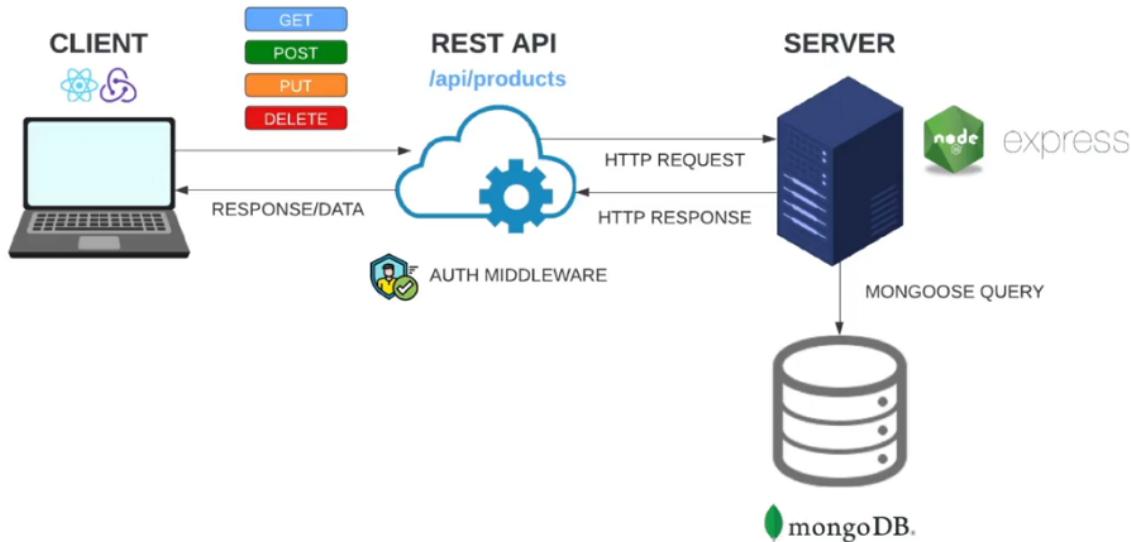
تصویر ۸ - کامپونت ProductScreen

۲. فصل دوم - ارائه و دریافت داده

۱-۱. معماری پروژه

در این فصل قرار است به پیاده سازی بخش سرور سایت بپردازیم اما قبل از آن خوب است که یک نگاه اجمالی به معماری پروژه بیندازیم.

Full Stack Workflow



تصویر ۹ - Workflow

در این پروژه قرار است از تکنولوژی MERN Stack استفاده کنیم. به اینصورت که بخش فرانت‌اند پروژه را با کتابخانه React پیاده سازی می‌کنیم. سرور سایت را با استفاده از فریمورک MongoDB و Node.js و Express استفاده خواهیم کرد و از پایگاه غیر رابطه ای MongoDB استفاده خواهیم کرد. معماری پروژه هم بدین صورت است که بخش فرانت‌اند و سرور سایت ما به

وسیله پروتکل REST API با یکدیگر ارتباط دارند. به اینصورت که بخش فرانت‌اند زمانی که اطلاعاتی را نیاز دارد به سمت سرور درخواست ارسال می‌کند و سرور اطلاعات مورد نظر را به سمت فرانت‌اند برمی‌گرداند و فرانت‌اند آن را نمایش می‌دهد. راجع به مفاهیم دیگر مثل ODM نیز در فصل های آتی صحبت خواهد شد.

۲-۲. سرور Express

در اینجا شروع به درست کردن اولین بخش از سرور می‌کنیم. فایل package.json که سند Node پروژه هایی است که با Node پیاده سازی شده است را به وسیله تعریف یک پروژه Base ایجاد می‌کنیم. ازین پس پکیج‌ها و اسکریپت‌های پروژه را درون این فایل کنترل می‌کنیم.

```

...
backend > JS server.js > ...
1 import express from 'express';
2 const port = 5000;
3
4 const app = express();
5
6 app.get('/', (req, res) => {
7   res.send('API is running...');
8 });
9
10 app.listen(port, () => console.log(`Server running on port ${port}`));
11

```

تصویر ۱ - اولین سرور Express

حال اگر سرور را استارت بزنیم و به وسیله مرورگر به پورت ۵۰۰۰ برویم مشاهده می‌کنیم که سرور به درستی پیاده سازی شده است. حال یک دیتای پیش‌فرض به پروژه به اسم products اضافه می‌کنیم تا بتوانیم اولین API خود را بنویسیم.

۲-۳. اسکریپت های پروژه

ما نیاز داریم که یکسری اسکریپت برای پروژه طراحی کنیم که از تکرار یک سری دستورات جلوگیری کنیم. برای شروع این کار نیاز است دو پکیج Nodemon و Concurrently را به نیازمندی های بخش توسعه اضافه کنیم.

```
> Debug
"scripts": {
  "start": "node backend/server.js",
  "server": "nodemon backend/server.js",
  "client": "npm start --prefix frontend",
  "dev": "concurrently \"npm run server\" \"npm run client\""
},
```

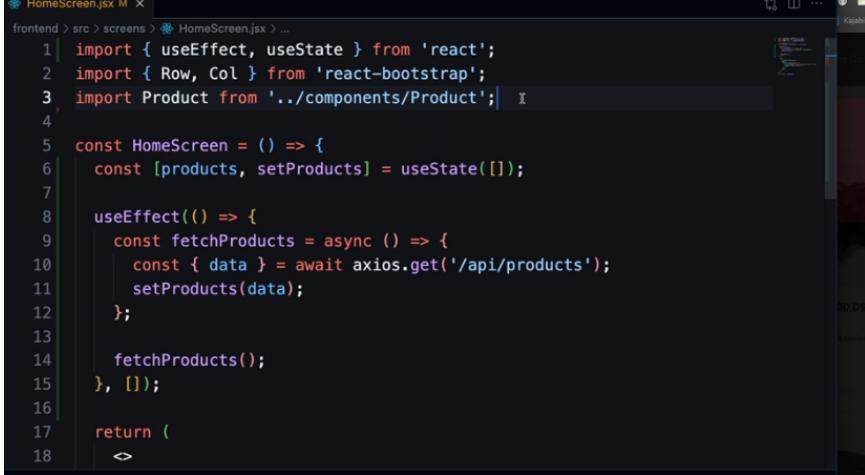
تصویر ۱۱ - اسکریپت ها

دستور start سرور را اجرا می کند. دستور server هم سرور را اجرا می کند اما تفاوتش با قبلی در این است که اگر هنگامی که سرور در حال اجرا باشد ما تغییری در کد ایجاد کنیم آن تغییر را می توانیم در همان زمان مشاهده کنیم. دستور client هم پروژه فرانت‌اند را اجرا می کند. اما دستور آخر و مهمترین دستور dev که پروژه فرانت‌اند و سرور را به طور همزمان اجرا می کند.

۴-۲. درخواست داده از سمت فرانت‌اند

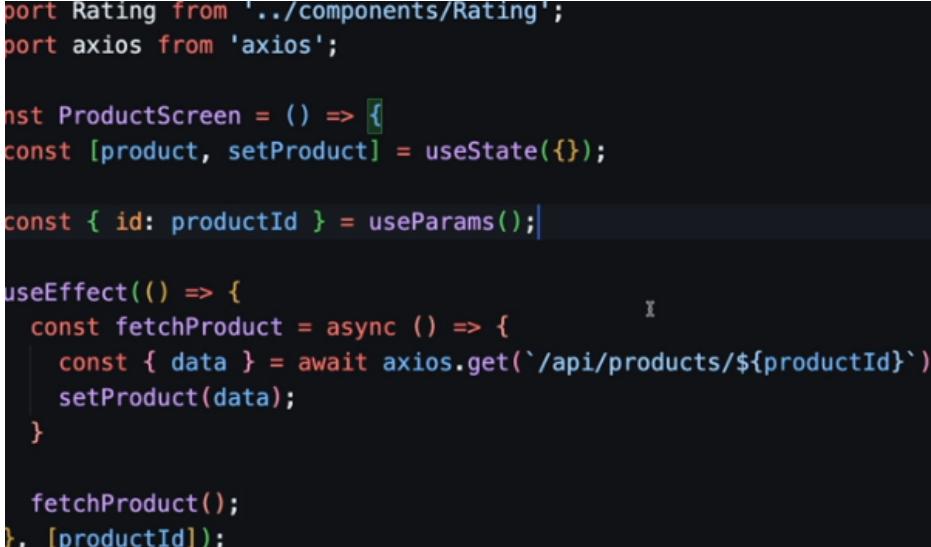
حال می خواهیم از سمت فرانت‌اند به سمت سرور درخواست بزنیم که محصولات را به ما برگرداند. برای این کار باید از کتابخانه AXIOS استفاده کنیم که بوسیله آن بتوانیم درخواست های HTTP خود را از سمت فرانت‌اند انجام دهیم. حال در کامپوننت ProductScreen و

اطلاعات را به وسیله درخواست زدن به سرور داینامیک می‌کنیم.



```
frontend > src > screens > HomeScreen.jsx > ...
1 import { useEffect, useState } from 'react';
2 import { Row, Col } from 'react-bootstrap';
3 import Product from '../components/Product';
4
5 const HomeScreen = () => {
6   const [products, setProducts] = useState([]);
7
8   useEffect(() => {
9     const fetchProducts = async () => {
10       const { data } = await axios.get('/api/products');
11       setProducts(data);
12     };
13
14     fetchProducts();
15   }, []);
16
17   return (
18     <>
```

تصویر ۱۲ - دریافت اطلاعات در صفحه اصلی



```
port Rating from '../components/Rating';
port axios from 'axios';

const ProductScreen = () => {
  const [product, setProduct] = useState({});

  const { id: productId } = useParams();

  useEffect(() => {
    const fetchProduct = async () => {
      const { data } = await axios.get(`/api/products/${productId}`);
      setProduct(data);
    };

    fetchProduct();
  }, [productId]);
```

تصویر ۱۳ - دریافت دیتا در صفحه جزئیات محصول

۳. فصل سوم - پایگاه داده

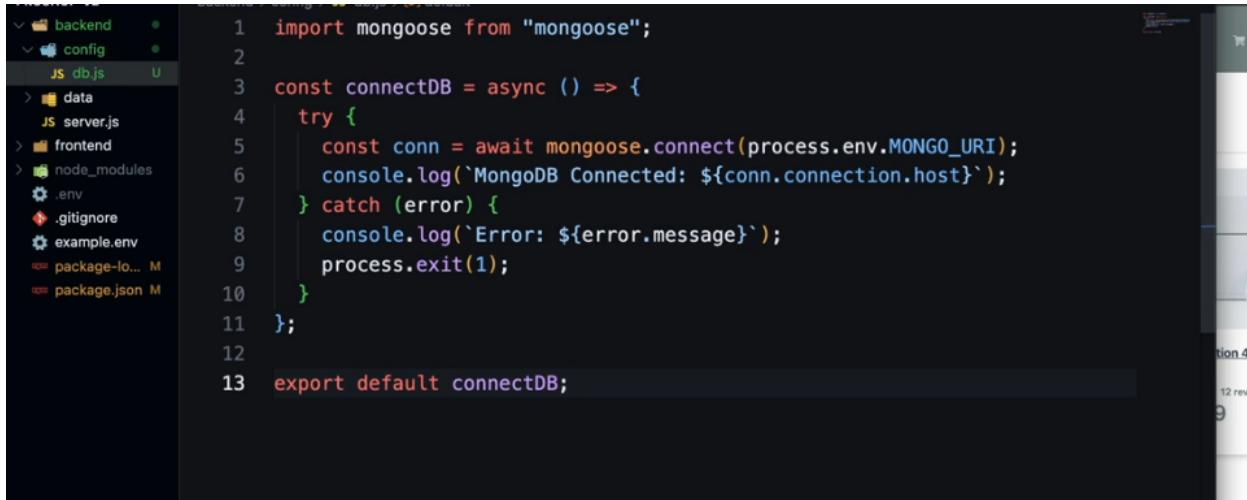
۱-۳. شروع کار با پایگاه داده

در این فصل قرار است پایگاه داده خود را تعریف کنیم و قرار است از MongoDB که یک پایگاه داده غیر رابطه ای یا NoSQL است استفاده کنیم. پایگاههای داده NoSQL مزیت‌های بسیار زیادی دارند که آن‌ها را برای سیستم‌های بزرگ و توزیع شده تبدیل به بهترین گزینه می‌کند. حال نیاز داریم برای تعریف پایگاه داده خود از یک سرور و یا سرویس ابری تحت وب استفاده کنیم که خود MongoDB برای پروژه‌های کوچک یک سرویس ابری رایگان به نام MongoDB Atlas می‌باشد. نحوه کار هم به اینصورت است که به شما یک آدرس URI داده می‌شود که شما می‌توانید آن را به عنوان آدرس پایگاه داده خود قرار دهید. حال نیاز به یک محیط گرافیکی خوب برای ارتباط مستقیم با پایگاه داده داریم که از MongoDB Compass برای اینکار استفاده می‌کنیم. آدرس URI ای که داریم را درون آن وارد می‌کنیم و به اینصورت با موفقیت به پایگاه داده خود متصل می‌شویم.

۲-۳. ارتباط با پایگاه داده از طریق برنامه

حال نیاز است از درون برنامه با پایگاه داده ارتباط بگیریم که نیاز است از یک رابط برای کار کردن با پایگاه داده خود استفاده کنیم. پکیج Mongoose را روی پروژه نصب می‌کنیم. یک چهارچوب محبوب و قوی برای استفاده روی پایگاه داده MongoDB است. با استفاده از چهارچوب Mongoose شما می‌توانید بر مبنای پایگاه‌داده MongoDB داده‌ها را استفاده، فراخوانی و ذخیره کنید. ابتدا در شروع یک پوشه config در پوشه پروژه ایجاد می‌کنیم و فایلی به نام db را درون آن قرار می‌دهیم که تابعی برای ارتباط با پایگاه داده‌م درون آن تعریف

می‌کنیم. حال اینجا نیاز است یک فایل به نام env درون پروژه‌مان ایجاد کنیم که تمامی متغیرهای محلی پروژه‌مان از جمله پورت و URI پایگاه داده را درون آن قرار دهیم تا زمانی که می‌خواهیم آن‌ها را روی سرور عمومی دیپللوی کنیم این اطلاعات قابل دسترسی نباشند. در انتهای تابع تعریف شده را درون فایل سرور فراخوانی می‌کنیم.



```

1 import mongoose from "mongoose";
2
3 const connectDB = async () => {
4   try {
5     const conn = await mongoose.connect(process.env.MONGO_URI);
6     console.log(`MongoDB Connected: ${conn.connection.host}`);
7   } catch (error) {
8     console.log(`Error: ${error.message}`);
9     process.exit(1);
10 }
11 };
12
13 export default connectDB;

```

تصویر ۱۳ - فایل پایگاه داده

۳-۳. مدل کردن داده‌ها

حال که پایگاه داده خود را ایجاد کردیم نیاز است که داده‌های مورد نیاز برای پروژه را مدل کنیم. برای این پروژه نیاز است سه مدل برای محصولات و کاربران و سفارشات خود داشته باشیم. یک پوشه جدید به نام model درون پروژه ایجاد می‌کنیم و برای هر کدام از مدل‌های گفته شده یک فایل جداگانه ایجاد می‌کنیم. یک فایل به نام seeder هم درون پروژه اضافه می‌کنیم که بوسیله آن بتوانیم داده‌های پیش فرضی را برای ایمپورت کردن درون پروژه بر اساس مدل‌های MongoDB ایجاد کردیم داشته باشیم. یکبار داده‌ها را ایمپورت می‌کنیم و خروجی را درون Compass مشاهده می‌کنیم.

داده های مورد نیاز برای محصولات: نام، عکس، نام برنده، کتگوری، توضیحات، امتیاز، تعداد بازخورد، قیمت، تعداد باقیمانده

داده های مورد نیاز برای کاربران: نام، ایمیل، پسورد، ادمین بودن

proshop.products

9 DOCUMENTS

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find ⌂

[ADD DATA](#) [EXPORT DATA](#) 1 - 9 of 9 ⌂ ⌂ ⌂ ⌂ ⌂

```
_id: ObjectId('6568878464a1d06bbd36882c')
user: ObjectId('6568878364a1d06bbd368828')
name: "Sample dress 1"
image: "/images/1.jpg"
brand: "Proshop"
category: "Clothes"
description: "The description of Sample dress 1"
rating: 4.5
numReviews: 12
price: 89.99
countInStock: 10
reviews: Array (empty)
__v: 0
createdAt: 2023-11-30T13:00:52.104+00:00
updatedAt: 2023-11-30T13:00:52.104+00:00
```

تصویر ۱۴ - نمونه دیتا محصولات

The screenshot shows the MongoDB Compass interface with the database 'proshop' and collection 'users'. There are 3 documents and 2 indexes. The 'Documents' tab is selected. A search bar at the top says 'Type a query: { field: 'value' } or Generate query'. Below it are buttons for 'ADD DATA', 'EXPORT DATA', 'Explain', 'Reset', 'Find', and 'Options'. The results show two documents:

```

_id: ObjectId('6568878364a1d06bbd368829')
name: "Mahan Jafari"
email: "mahan@email.com"
password: "$2a$10$PivYfS/sn2UhBr6Wm9rQmuffF0pgcO9mIzsraU3S9s.fCzCJThLKTm"
isAdmin: false
__v: 0
createdAt: 2023-11-30T13:00:51.912+00:00
updatedAt: 2023-11-30T13:00:51.912+00:00

_id: ObjectId('6568878364a1d06bbd36882a')
name: "Peyman Jafari"
email: "peyman@email.com"
password: "$2a$10$9It35s0/qQXlAFpdyi/kAeQ05BsyvmqlAZ8K"
isAdmin: false
__v: 0
createdAt: 2023-11-30T13:00:51.913+00:00
updatedAt: 2023-11-30T13:00:51.913+00:00

```

تصویر ۱۵ - نمونه داده کاربران

۳-۴. دریافت محصولات

حال میخواهیم اولین Router خود در سمت سرور که محصولات را به ما برمیگرداند را پیاده سازی کنیم. برای این کار نیاز است یک پوشه به نام routes ایجاد کنیم و یک فایل برای محصولات ایجاد کنیم. دو اندپوینت درون این فایل قرار می‌گیرند یکی آرایه ای از محصولات را برمیگرداند و دیگری با وارد کردن آیدی محصول آن محصول مورد نظر را به ما برمیگرداند. یک تابع به نام asyncHandler نیز پیاده سازی می‌کنیم که بتوانیم درخواست های ناهمزمان به سمت پایگاه ارسال نماییم.

```
kend > routes > JS productRoutes.js > ⚡ asyncHandler() callback > 🌐 product
5
6 router.get('/', asyncHandler(async (req, res) => {
7   const products = await Product.find({}); 
8   res.json(products);
9 });
10
11 router.get('/:id', asyncHandler(async (req, res) => {
12   const product = await Product.findById(req.params.id);
13   res.json(product);
14 });
15
16 export default router;
17
```

تصویر ۱۶ Router - محصولات

۳-۵. بهبود پروژه

برای بهتر شدن دو کدی که نوشته شده است نیاز است دو ویژگی دیگر نیز به پروژه اضافه کنیم. اولین بخش اضافه کردن ارور است به اینصورت که زمانی که درخواست نادرستی به اندپوینت هایی که پیاده سازی کردیم وارد شد بتواند پیغام خطای مناسبی به ما نشان دهد. برای اینکار یک middleware پیاده سازی میکنیم و اندپوینت های نوشته شده را به وسیله آن بازسازی میکنیم. دومین ویژگی خوبی که میتوانیم به پروژه اضافه کنیم این است که درون پوشه routes با پایگاه داده خود به صورت مستقیم ارتباط نگیریم و به جای آن یک پوشه به نام controllers تعریف کنیم که تمامی توابعی که نیاز داریم برای ارتباط گرفتن با پایگاه داده را در آن پیاده سازی کنیم و آن ها را در مورد نظرشان صدا بزنیم.

۴. فصل چهارم - مدیریت State ها

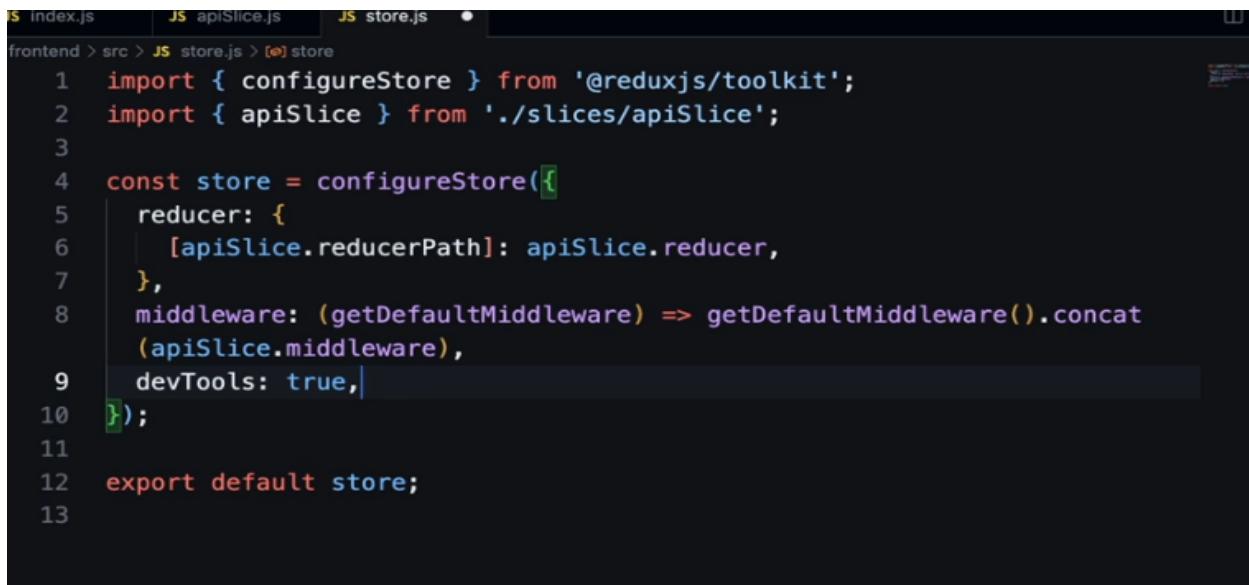
۱-۱. مفهوم اولیه و آشنایی با Redux

در این فصل قرار است به پیاده سازی State ها بپردازیم. مدیریت State یک جنبه مهم در توسعه نرم افزار است و به کنترل و ذخیره State (اطلاعات) یک اپلیکیشن در طول زمان می پردازد. مدیریت State به کنترل و مدیریت اطلاعات مختلفی که در یک اپلیکیشن وجود دارد می پردازد. این اطلاعات می توانند شامل وضعیت های کاربر، وضعیت های رابط کاربری، اطلاعات ورودی کاربر و سایر موارد مرتبط با عملکرد اپلیکیشن باشند. برای اینکار در این پروژه ما از کتابخانه Redux استفاده می کنیم. کتابخانه مدیریت State است که برای توسعه و مدیریت State اپلیکیشن های وب با استفاده از React یا دیگر کتابخانه های واسط کاربری طراحی شده است. در Redux، اطلاعات کل اپلیکیشن در یک store ذخیره می شود. این store یک نقطه مرکز برای ذخیره و مدیریت State است که از تمامی قسمت های اپلیکیشن قابل دسترسی است.

۱-۲ . تعریف ریداکس

ابتدا برای تعریف Redux در پروژه نیاز است دو پکیج react-redux و redux-toolkit کنیم. سپس store خود را تعریف می کنیم و آن را به دور کامپوننت App که کلیه پروژه فرانت اند ما درون آن قرار دارد قرار می دهیم. فایل کانفیگ store را می توانید در تصویر ۱۷ مشاهده

کنید. درون فایل store میتوانیم slice هایمان را قرار دهیم slice. مفهومی در حوزه مدیریت State در Redux است. برنامه در صورت درختی از داده‌ها ذخیره می‌شود که هر بخش از این درخت‌ها به عنوان یک slice شناخته می‌شود.



```
index.js      JS apiSlice.js      JS store.js •
frontend > src > JS store.js > [e] store
1 import { configureStore } from '@reduxjs/toolkit';
2 import { apiSlice } from './slices/apiSlice';
3
4 const store = configureStore({
5   reducer: {
6     [apiSlice.reducerPath]: apiSlice.reducer,
7   },
8   middleware: getDefaultMiddleware => getDefaultMiddleware().concat(
9     apiSlice.middleware),
10  devTools: true,
11);
12 export default store;
13
```

تصویر ۱۷ - فایل store

۴-۳ Slice . محصولات

حال میخواهیم تمامی محصولات را درون یک slice قرار دهیم که بتوانیم همه محصولات را روی یک State عمومی داشته باشیم که بتوانیم از همه کامپوننت‌ها به آن دسترسی داشته باشیم. یک پوشه slices درون پوشه frontend ایجاد می‌کنیم. ابتدا یک slice برای تمامی اندپوینت‌ها تعریف می‌کنیم و بر اساس آن یک productsApislice ایجاد می‌کنیم و آنجاهايی که د محصولات پیاده سازی شده بود داده‌ها را بوسیله این API بازنویسی یا به اصطلاح dispatch می‌کنیم.

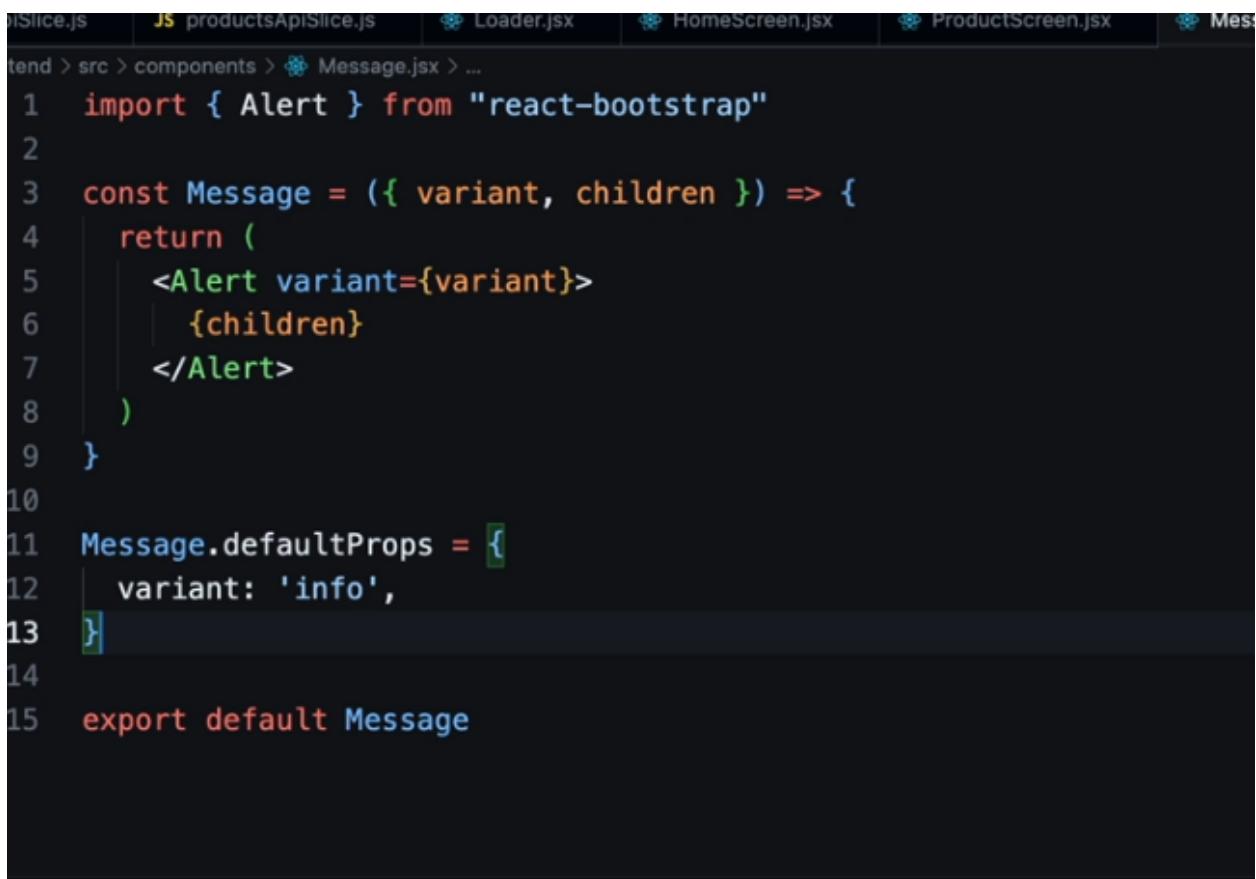
```
2 import { apiSlice } from "./apiSlice";
3
4 export const productsApiSlice = apiSlice.injectEndpoints({
5   endpoints: (builder) => ({
6     getProducts: builder.query({
7       query: () => ({
8         url: PRODUCTS_URL,
9       }),
10      keepUnusedDataFor: 5
11    })
12  }),
13});
14
```

تصویر ۱۸ - اسلایس محصولات

این کار را برای صفحه جزئیات محصول نیز انجام می‌دهیم تا دیتای همه صفحات از یک منبع دریافت شود.

۴-۴ . کامپونت Loading و Message

حال نیاز است دو کامپونت جدید به پروژه اضافه کنیم. کامپونت Message که از آن برای نمایش پیام هایی که میخواهیم به کاربر نمایش دهیم استفاده میکنیم و کامپونت Loading که برای زمان هایی مانند زمانی که میخواهیم API را از سمت سرور دریافت کنیم از آن استفاده میکنیم. از react-bootstrap برای اینکار کمک میگیریم.



```
JSlice.js | JS productsApiSlice.js | Loader.jsx | HomeScreen.jsx | ProductScreen.jsx | Message.js
tend > src > components > Message.jsx > ...
1 import { Alert } from "react-bootstrap"
2
3 const Message = ({ variant, children }) => {
4   return (
5     <Alert variant={variant}>
6       {children}
7     </Alert>
8   )
9 }
10
11 Message.defaultProps = {
12   variant: 'info',
13 }
14
15 export default Message
```

تصویر ۱۹ - کامپونت Message

```

EXPLORER      ...
PROSHOP-V2
> backend
< frontend
> node_modules
> public
< src
> assets
< components
  < Footer.jsx
  < Header.jsx
  < Loader.jsx
  < Product.jsx
  < Rating.jsx
< screens
  < HomeScreen...
  < ProductScre...
< slices
  JS apiSlice.js
  JS productsApi...
JS App.js
JS constants.js
JS index.js
JS reportWebVit...
JS loader.jsx

JS apiSlice.js   JS productsApiSlice.js   JS Loader.jsx   JS ProductScreen.jsx

frontend > src > components > Loader.jsx > Loader
1 import { Spinner } from "react-bootstrap";
2
3 const Loader = () => {
4   return (
5     <Spinner
6       animation="border"
7       role="status"
8       style={{
9         width: "100px",
10        height: "100px",
11        margin: "auto",
12        display: "block",
13      }}
14     ></Spinner>
15   )
16 }
17 export default Loader

```

تصویر ۲ - کامپونت Loader

۵. فصل پنجم - سبد خرید

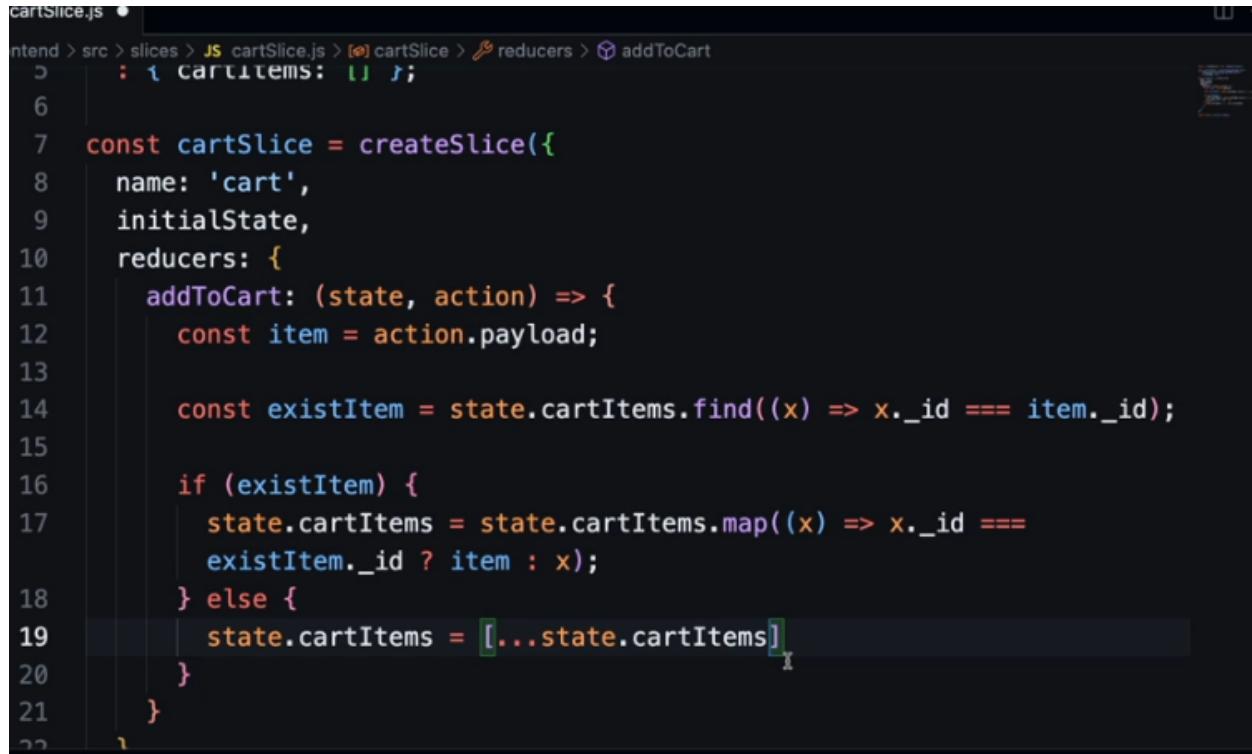
۱-۵. شروع کار با سبد خرید

در این فصل قرار است سبد خرید پیاده سازی شود. برای شروع باید یک slice جدید در ریداکس به نام cartSlice ایجاد کنیم که داده های سبد خرید را ابتدا از local storage می خواند و اگر این داده وجود نداشت به جای آن یک آرایه خالی قرار می دهد.

۲-۵. تابع اضافه شدن به سبد خرید

حال نیاز است یک reducer برای cartSlice پیاده سازی کنیم که عمل اضافه شدن به سبد خرید را انجام دهد و اطلاعاتی از قبیل قیمت، مالیات، هزینه شیپینگ و هزینه کلی را به ما

برگرداند. حال باید این تابع را زمانی که دکمه اضافه شدن به سبد زده می‌شود پیاده شود به اینصورت که داده‌های مربوط به محصول را به آرایه سبد خرید ما اضافه کند و وارد صفحه سبد خرید شود.



```

cartSlice.js ●
intend > src > slices > cartSlice.js > reducers > addCart
  : 1 cartItems: [ ];
  6
  7 const cartSlice = createSlice({
  8   name: 'cart',
  9   initialState,
 10  reducers: {
 11    addCart: (state, action) => {
 12      const item = action.payload;
 13
 14      const existItem = state.cartItems.find((x) => x._id === item._id);
 15
 16      if (existItem) {
 17        state.cartItems = state.cartItems.map((x) => x._id ===
 18          existItem._id ? item : x);
 19      } else {
 20        state.cartItems = [...state.cartItems]
 21      }
 22    }
 23  }

```

تصویر ۲۱ - cartSlice

۳-۵. نمایش تعداد آیتم سبد خرید

حال می‌خواهیم تعداد آیتمی که کاربر در سبد خرید قرار می‌دهد را در کامپوننت Header نمایش دهیم. برای اینکار از کامپوننت Badge در react-bootstrap استفاده می‌کنیم. به وسیله useSelector دیتا‌های store را از درون cartItems دریافت می‌کنیم و به وسیله آن تعداد آیتم درون سبد را نمایش می‌دهیم.

```
<LinkContainer to='/cart'>
  <Nav.Link>
    <FaShoppingCart /> Cart
    {
      cartItems.length > 0 && (
        <Badge pill bg='success' style={{marginLeft: '5px'}}>
          { cartItems.reduce((a, c) => a + c.qty, 0)
        </Badge>
      )
    }
  </Nav.Link>
</LinkContainer>
```

تصویر ۲۲ - نمایش Badge

۵-۴. صفحه سبد خرید

حال به پیاده سازی صفحه سبد خرید می پردازیم که دارای ویژگی هایی نظیر نمایش آیتم ها، عوض کردن تعداد، نمایش قیمت و دکمه رفتن به صفحه پرداخت می باشد. کامپوننتی تحت عنوان CartScreen درون پوشه screens ایجاد می کنیم و ویژگی هایی را که گفتیم در این کامپوننت پیاده سازی می کنیم. خروجی کار به صورت زیر است:

The screenshot shows a shopping cart page. At the top, there's a header with the ProShop logo, a search bar containing 'Search Products...', and a navigation bar with 'Cart 1', 'Admin User', and 'Admin'. Below the header, the title 'Shopping Cart' is displayed. The main content area shows a single item: 'Sample dress 3' with a price of '\$129.99'. There are quantity controls (a minus button, a '1' input field, and a plus button) and a delete icon. To the right, a box titled 'Subtotal (1) items' shows the total '\$129.99' and a 'Proceed To Checkout' button.

تصویر ۲۳ - خروجی

۵-۵. حذف از سبد خرید

تابع حذف از سبد خرید را همانند تابع اضافه شدن به سبد خرید پیاده سازی می‌کنیم به اینصورت که یک slice به نام `removeFromCart` روی `cartSlices` پیاده سازی می‌کنیم و در صفحه `Cart` نیز یک تابع هندرل برای آن پیاده سازی می‌کنیم.

```
23     }
24
25     return updateCart(state);
26   },
27   removeFromCart: (state, action) => {
28     state.cartItems = state.cartItems.filter((x) => x._id !== action.payload);
29
30     return updateCart(state);
31   },
32 },
33 );
34
35 export const { addToCart, removeFromCart } = cartSlice.actions;
36
37 export default cartSlice.reducer;
38
```

تصویر ۲۴ - حذف از سبد

۶. فصل ششم - ویژگی های اضافه

۱-۶. پیاده سازی بخش بازخورد

در این فصل به تکمیل ویژگی های اپلیکیشن می پردازیم. اولین ویژگی بخش بازخورد هر محصول است. هر بخش بازخورد از ترکیب چند ویژگی نام، امتیاز، کامنت، نام کاربر تشکیل می شود که باید در بخش productHandler به پیاده سازی آن بپردازیم.تابع createProductReview را ایجاد می کنیم و ویژگی های گفته شده را در آن پیاده سازی می کنیم.

```
productController.js •
```

```
| > controllers > JS productController.js > [e] createdProductReview > [o] asyncHandler() callback > [o] alreadyReviewed > [o] product.reviews.find() c
```

```
const product = await Product.findById(req.params.id);
```

```
if (product) {
```

```
    const alreadyReviewed = product.reviews.find(
        (review) => review.user.toString() === req.user._id.toString()
    );
```

```

        if (alreadyReviewed) {
            res.status(400);
            throw new Error('Product already reviewed');
        }
    }
```

```
const review = {
    name: req.user.name,
    rating: Number(rating),
    comment,
    user: req.user._id,
};
```

تصویر ۲۵ - بخش Review

حال به پیاده سازی بخش فرانت‌اند آن می‌پردازیم. ابتدا نیاز است بازخورد هایی را که از سمت سرور دریافت می‌کنیم درون `sote` ذخیره کنیم سپس به وسیله کامپونت `ListGroup` داده های مربوطه را نمایش می‌دهیم. خروجی کار به صورت زیر است:

Reviews

John Smith
 2023-04-26

This is a great product

Write a Customer Review

Rating

Select...

Comment

Submit

تصویر ۲۶ - بخش خروجی

۶-۲. پیاده سازی بخش صفحه‌بندی

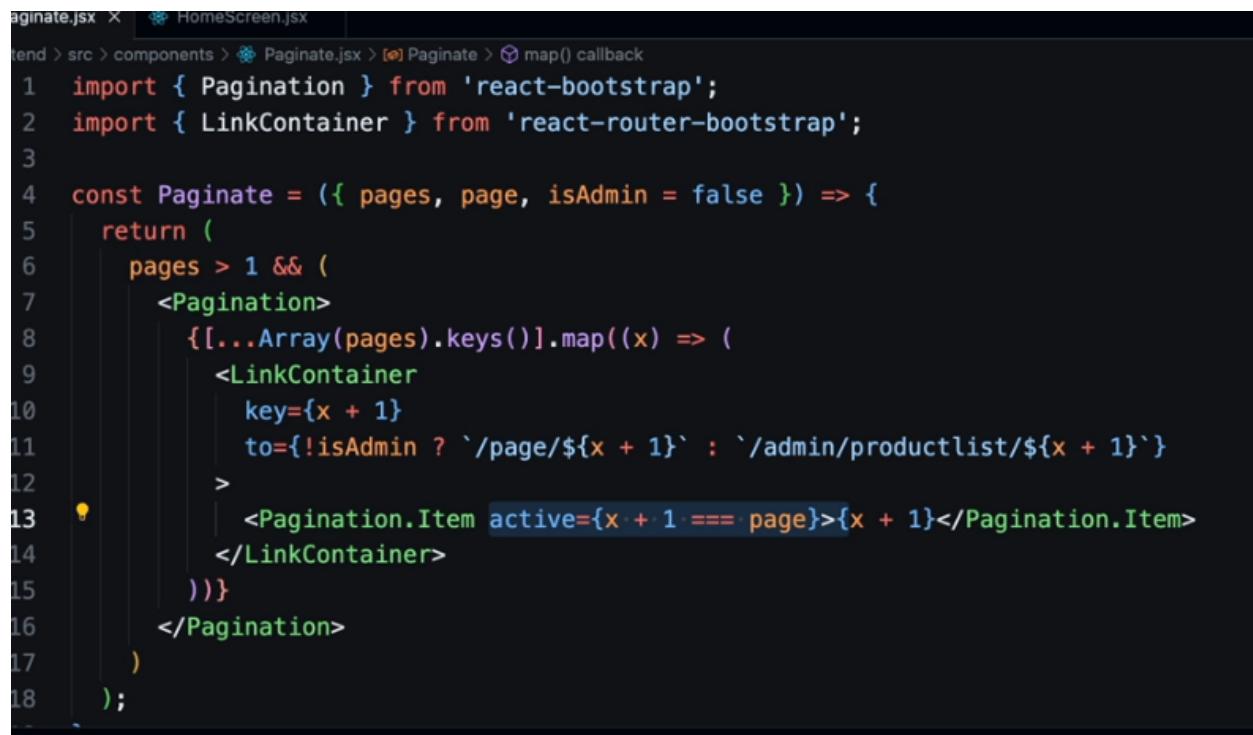
در این بخش به پیاده سازی صفحه بندی محصولات می‌پردازیم. به اینصورت که باید در بخش routing روتر جدیدی برای صفحات ایجاد کنیم و همچنین در بخش productHandler زمانی که محصولات دریافت می‌شوند باید تابع زیر پیاده سازی شود:

```
const getProducts = asyncHandler(async (req, res) => {
  const pageSize = 2;
  const page = Number(req.query.pageNumber) || 1;
  const count = await Product.countDocuments();

  const products = await Product.find({})
    .limit(pageSize)
    .skip(pageSize * (page - 1));
  res.json({products, page, pages: Math.ceil(count / pageSize)});
});
```

تصویر ۲۷ - بخش صفحه بندی

حال نیاز است در سمت فراتاند کامپوننتی به نام Paginate ایجاد کنیم که کار صفحه‌بندی محصولات را انجام دهد.



```
paginate.jsx X HomeScreen.jsx
tend > src > components > paginate.js > Paginate > map() callback
1 import { Pagination } from 'react-bootstrap';
2 import { LinkContainer } from 'react-router-bootstrap';
3
4 const Paginate = ({ pages, page, isAdmin = false }) => {
5   return (
6     pages > 1 && (
7       <Pagination>
8         {[...Array(pages).keys()].map((x) => (
9           <LinkContainer
10             key={x + 1}
11             to={!isAdmin ? `/page/${x + 1}` : `/admin/productlist/${x + 1}`}
12           >
13             <Pagination.Item active={x + 1 === page}>{x + 1}</Pagination.Item>
14           </LinkContainer>
15         )));
16       </Pagination>
17     )
18   );
}
```

تصویر ۲۸ - کامپوننت Paginate

۳-۶. پیاده سازی اسلایدر

در این بخش در صفحه اصلی محصولات یک اسلایدر برای نمایش محصولات مهم ایجاد می‌کنیم. کامپوننتی به نام ProductCarousel ایجاد می‌کنیم و به وسیله کامپوننت Carousel در react-bootstrap این کامپوننت را ایجاد می‌کنیم. داده‌های مورد نیاز را هم از همان API محصولات دریافت می‌کنیم.

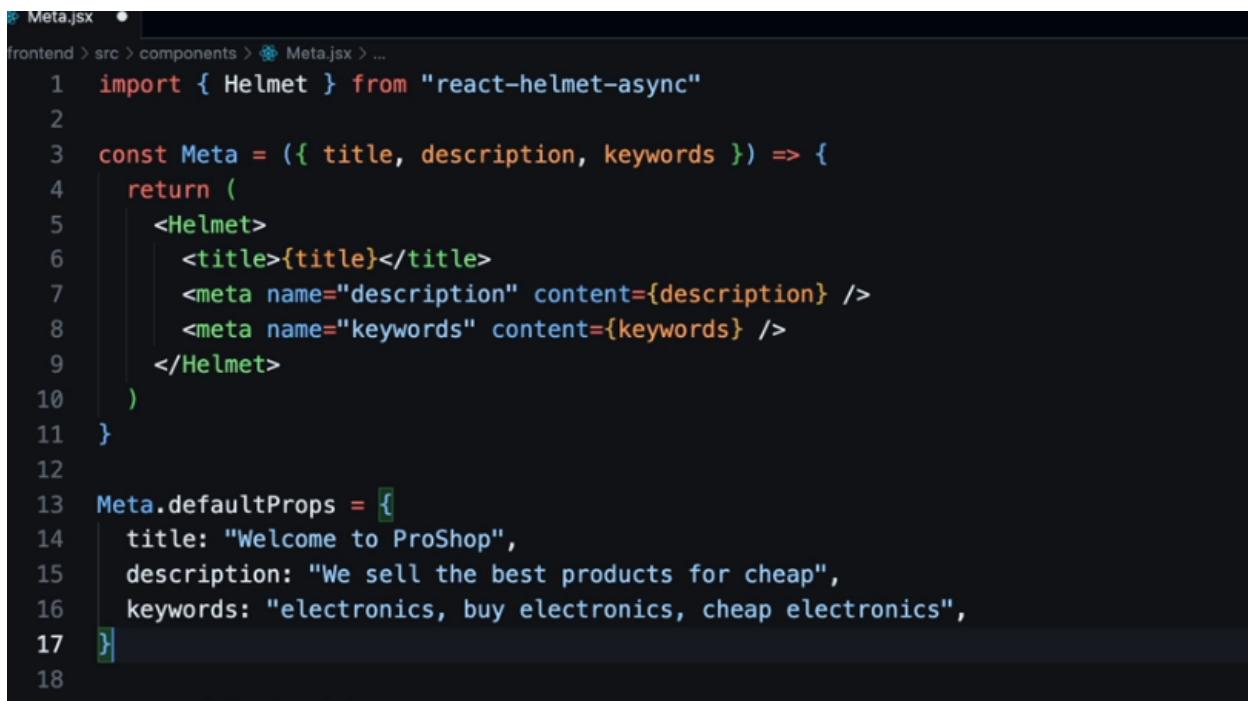
A screenshot of a code editor showing the `ProductCarousel.jsx` component. The code uses the `Carousel` and `Carousel.Item` components from a library, mapping over a `products` array to render each product's name and price in an `##` element. An `Loader` component is used for loading state, and a `Message` component is used for error handling.

```
ProductCarousel.jsx X HomeScreen.jsx
frontend > src > components > ProductCarousel.jsx > ProductCarousel > products.map() callback
11     <Loader />
12   ) : error ? (
13     <Message variant='danger'>{error}</Message>
14   ) : (
15     <Carousel pause='hover' className='bg-primary mb-4'>
16       {products.map((product) => (
17         <Carousel.Item key={product._id}>
18           <Link to={`/product/${product._id}`}>
19             <Image src={product.image} alt={product.name} fluid />
20             <Carousel.Caption className='carousel-caption'>
21               <h2>
22                 {product.name} (${product.price})
23               </h2>
24             </Carousel.Caption>
25           </Link>
26         </Carousel.Item>
27       ))}
28     </Carousel>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL node +
```

تصویر ۲۹ - کامپوننت ProductCarousel

۶-۴. داده های Meta

در تمامی صفحات دارای داده های Meta شبیه به هم می باشند. برای اینکه بتوانیم داده های Meta هر صفحه را شخصی سازی کنیم باید پکیج `react-helmet-async` را نصب کنیم. یک کامپوننت Meta ایجاد می کنیم و به وسیله آن در هر صفحه داده های Meta را شخصی سازی می کنیم.

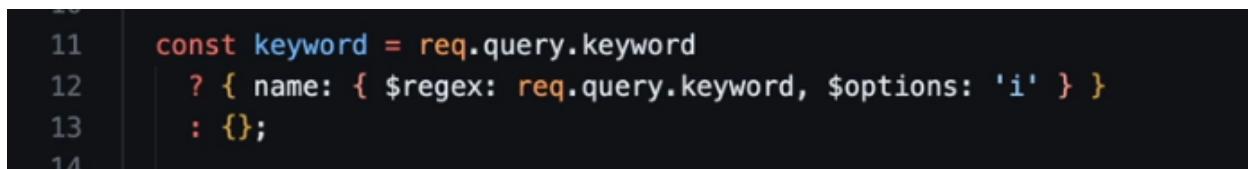


```
Meta.jsx
frontend > src > components > Meta.jsx > ...
1 import { Helmet } from "react-helmet-async"
2
3 const Meta = ({ title, description, keywords }) => {
4   return (
5     <Helmet>
6       <title>{title}</title>
7       <meta name="description" content={description} />
8       <meta name="keywords" content={keywords} />
9     </Helmet>
10   )
11 }
12
13 Meta.defaultProps = {
14   title: "Welcome to ProShop",
15   description: "We sell the best products for cheap",
16   keywords: "electronics, buy electronics, cheap electronics",
17 }
18
```

تصویر ۳ - کامپوننت Meta

۶-۵. پیاده سازی بخش جستجو

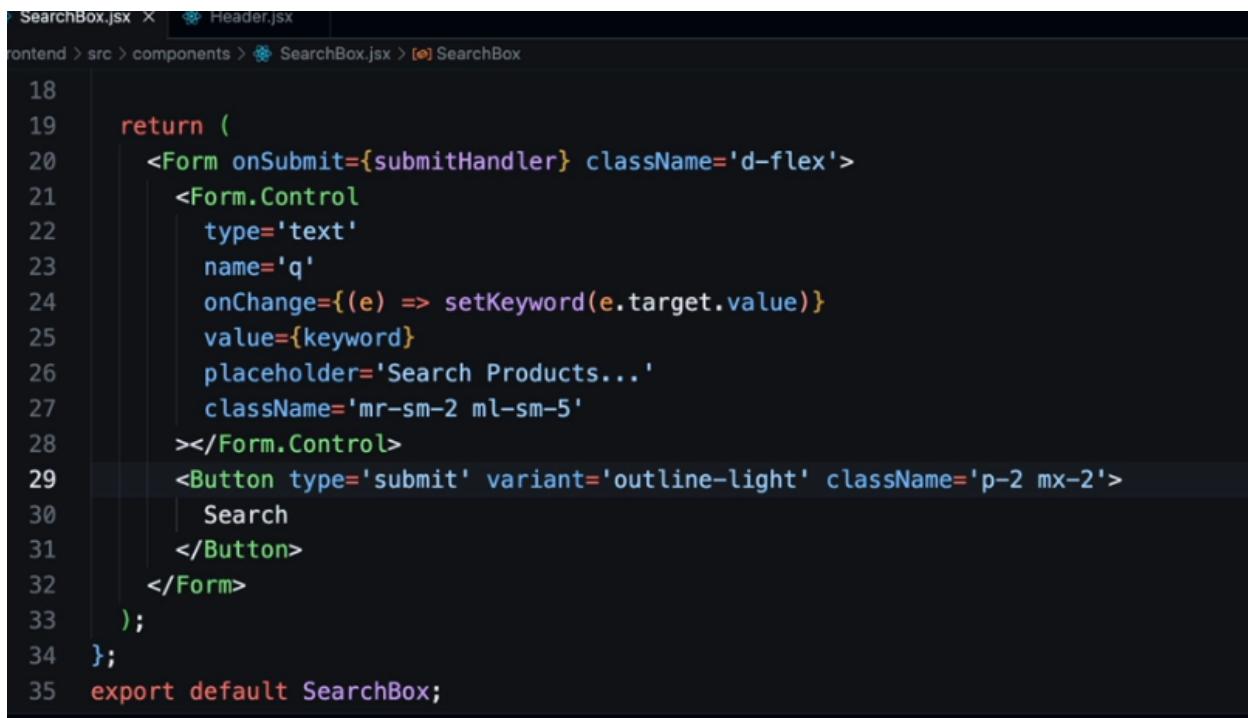
در این بخش می خواهیم یک باکس جستجو ایجاد کنیم برای اینکه کاربر بتواند محصول مورد نظر خود را جستجو کند. ابتدا یک Router جدید ایجاد می کنیم که بر اساس ورودی کاربر بتواند محصولات ما را فیلتر کند. برای اینکار باید از regEx کمک بگیریم.



```
11 const keyword = req.query.keyword
12 ? { name: { $regex: req.query.keyword, $options: 'i' } }
13 : {};
```

تصویر ۴ - سرچ keyword

حال نیاز است یک کامپوننت SearchBox ایجاد کنیم که ورودی را از کاربر می‌گیرد و یک تابع submitHandler برای آن ایجاد می‌کنیم که کار فیلتر کردن داده‌ها را انجام دهد.



The screenshot shows a code editor with the file 'SearchBox.jsx' open. The code defines a functional component 'SearchBox'. It uses the 'Form' and 'Control' components from a library. The search input has a placeholder 'Search Products...', a value prop set to 'keyword', and an onChange handler that sets 'keyword' to the input value. The form has a submit handler 'submitHandler'. A button labeled 'Search' triggers the submission. The component is styled with 'd-flex' and 'mr-sm-2 ml-sm-5' classes. The code ends with an export statement.

```
18
19     return (
20       <Form onSubmit={submitHandler} className='d-flex'>
21         <Form.Control
22           type='text'
23           name='q'
24           onChange={(e) => setKeyword(e.target.value)}
25           value={keyword}
26           placeholder='Search Products...'
27           className='mr-sm-2 ml-sm-5'
28         ></Form.Control>
29         <Button type='submit' variant='outline-light' className='p-2 mx-2'>
30           Search
31         </Button>
32       </Form>
33     );
34   };
35   export default SearchBox;
```

تصویر ۳۲ - کامپوننت SearchBox

منابع

[1] آموزش MERN از پایه:

<https://www.traversymedia.com/mern-stack-from-scratch>

[2] داکیومنت React Bootstrap:

<https://react-bootstrap.netlify.app/>

[3] داکیومنت Redux Toolkit:

<https://redux-toolkit.js.org/>

[4] داکیومنت React:

<https://react.dev/>

[5] داکیومنت Express:

<https://expressjs.com/>