npx create-react-app blog-frontend

package.json

react

react-dom

react-scripts

Scripts.

public > index.html

<title> ... <title>

Delete logo, setupTests, App.test

App.js

x import logo.

x import App.css

Delete everything in index.css

src > components ← create.

- Header.js
- Footer.js

} include both in App.js

React Boobtrap UI library

Ex. material UI

Bootswatch — themes for bootstrap.

↳ Download any ex (bootstrap.min.css)

Terminal > frontend >

```
npm i react-bootstrap
```

App.js

```
import { Container } from 'react-bootstrap'

return (
  <>
    <Header />
      <main>
        <Container>
          <h1> Welcome </h1>
        </Container>
      </main>
    <Footer />
  </>
)
```

Search: React bootstrap navbar

→ Copy Navbar code → customise

Copy fontawesome library to public > index.html.

Add icon to cart menu.

Copy 6 images to public folder (dummy).

Create products.js file src folder.

↳ contains product data in an array.

Create folder

src > screens

> HomeScreen.js

> rafce

```
import {Row, Col} from 'react-bootstrap'
:
return (
    <>
        <h1> Latest Products </h1>
            <Row>
                {products.map(product => (
                    <Col>
                        <h3> {product.name} </h3>
                    </Col>
            </Row>
```

} Customise styles.

Replace <h3> with <product> component ⟶ must be created.

components > Product.js

rafce!

```
import { Card } from 'react-bootstrap'
  :
                                    (props) ← include props as parameter
return(
   <Card className='my-3 p-3 rounded'>
      <a href={`/product/${props.product._id}`}>
      <Card.Img src={props.product.image} variant='top' />
      </a>

   <Card.Body>
      <a href={`/product/${props.  ... "    >
      <Card.Title as 'div'>
           <strong> {props.product.name} </strong>
      </Card.Title>
      </a>

  :
```

Rating Component

Components > Product.js

```
import Rating from './Rating'
```

(Replace rating component 'div' in Pricing.js)

```
<Card.Text as='div'>
    <Rating value={props.product.rating}
            text={`${product.numReviews} reviews`} />
</Card.Text>
```

Create Rating Component in components > Rating.js

rafce

```
const Rating = ({value, text}) => {
    return (
        [ Fill in along with (star) reviews - ]
        :
    )
}

Rating.defaultProps = {
    color: '#f8e825'
}
```

destructuring

```
Rating.propTypes = {
    value: Proptypes.number.isRequired,
    text: Proptypes.string.isRequired,
    color: Proptypes.string,
}
```

index.css

```
h3 {
    padding: 1rem 0;
}

main {
    min-height: 80vh;
}

.rating span {
    margin: 0.1rem;
}
```

Install React Developer Tools → chrome Extension

Backend

Create backend folder in project.

Copy products.js in backend > data > products.js

- project
  - backend
    - data
      - products.js.
    - frontend.

Create server.js in backend folder

```
const express = require('express')

const app = express()

app.listen(5000, console.log('server running on port 5000'))
```

Term > (backend) node server

... displays server runnig on port 5000

```
app.get('/', (req, res) => {
    res.send('API is running...')
})
```

Modify data > products.js

:

export default products ⟹ (to) module.exports = products

backend > server.js

```
Const products = require('./data/products')

(modify) app.get('/api/products', (req, res) => {

    res.json(products)

})

//Getting single product

app.get('/api/products/:id', (req, res) => {

    Const product = products.find((p) => p._id ===
                                        req.params.id)

    res.json(product)   (single)

})
```

Restart server and check

Term > backend > node server

(check) localhost:5000/api/products

/1

fetching products from backend to React frontend (useEffect)

Term > frontend > npm i axios

npm start

HomeScreen.js

Remove import products from '../products'

import {useState} from 'react'

useEffect

```
const HomeScreen = () => {

    const [products, setProducts] = useState([])

    useEffect( () => {

        const fetchProducts = async () => {

            const {data} = await axios.get('/api/products')

            setProducts(data)
        }

        fetchProducts()
    }, [])
```

---

*

⟹ Gives no products as it looks for localhost:3000/api/p...

* Need to add proxy in frontend > package.json.

```
"name":
"proxy": "http://127.0.0.1:5000"
```

Customising ProductScreen.js & using useEffect()

```
const ProductScreen = ({match}) => {

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

    useEffect(() => {

        const fetchProduct = async () => {

            const {data} = await axios.get(`/api/products/
                                            ${match.params.id}`)

            setProduct(data)
        }

        fetchProduct()

    }, [])
        ← match

    return (
        :

    )
```

import

Setting up nodemon and concurrently to run backend and frontend at the same time.

root folder (srisai)

Terminal > (srisai) npm init

↳ creates package-json in root

package.json

Terminal > (~~back~~ root) npm i express

npm i -D nodemon concurrently

concurrently → Two at once

nodemon → watch server

package.json

```
"scripts": {
    "start": "node backend/server",
    "server": "nodemon backend/server",
    "client": "npm start --prefix frontend"
}
```

check npm run server

Using concurrently

"scripts" : {

"dev" : " concurrently \"npm run server\"

\"npm run dict\" "

check

npm run dev. ⟶ runs on both l: 3000
l: 5000

Setting env variables

root > npm i dotenv

npm run server ←-- Only backend server to run

server.js

Const express = ...

→ const dotenv = require('dotenv')

Const pro...

dotenv.config()

(Create .env in root)

↳ NODE_ENV = development

PORT = 5000.

Restart server → npm run server

server.js

Const PORT = process.env.PORT || 5000.

modify app.listen(PORT, consok.log(`semer running in ${process.env.NODE_E

node on ${PORT})

Add .env to .gitignore

ES modules

node --version. <14.9 --- update

package-json

"main" : "server.js",
"type" : "module", ← Add this.

for files import (.js) is required in (ESM)

server.js

import express from 'express'

import dotenv from 'dotenv'

import products from './data/products.js' ← Add .js (if not shows module error)

change back to

In products.js file

:

export default products ← update

Mongodb Atlas — Cloud DB.

Mongodb compass — Desktop.

No sql

↳ collections of documents.

Similar to json.

Create account in Mongodb Atlas → Cloud version.

→ Build a cluster

→ JS — preferred.

→ free tier

→ Aws cloud.

→ Name it → create.

mongodb.com → Software → Compass. → Try Now

→ Windows → Download.

Cluster → Security → Database Access

→ Auth method

→ Password. → Add user.

Network Access → Allow access from anywhere. → Confirm

Clusters > Collections

Add My own Data.

→ Database Name → Srisai

Collection Name → Products

Clusters > Connect > Connect using Mongodb compass

→ I have M.dB comp → Copy string

Open MongodB compass

Paste copied string .

Change /test/ to srisai

change <password> to your own

Click Connect

Clusters > Collections > Add My own Data.

Srisai > Products

Copy connection string and paste in .env

MONGO_URI = mongodb+srv://..... (COM).

↑ replace db and pass /srisai)

Modelling Data in :

backend > models

- userModel.js
- productModel.js
- orderModel.js

userModel.js

```
import mongoose from 'mongoose'

const userSchema = mongoose.Schema({

    name : {
        type: String,
        required : true
    },
    email : {
        type : String,
        required : true,
        unique : true
    },
    password : {
        :
    }
    isAdmin : {
        type : Boolean,
        reuired : true,
        default : false
    }
}, {
    timestamps : true
})
```

(Contd...)

(Contd...)

```
const User = mongoose.model('User', userSchema)

export default User
```

Ug, ProductModel.js

```
import mongoose from 'mongoose'

const productSchema = mongoose.Schema(
  {
    user : {
        type : mongoose.Schema.Types.ObjectId,
        required : true,
        ref : 'User'
    },
    image : {
        type : String,
        required : true
    },
    brand : {
        type : string,
        required : true
    },
    category : {
        type : string,
        required : true
    },
    description : {
        (similar to category - copy).
    },
```

(Contd...)

```
const reviewSchema = mongoose.Schema({
    name : { type : String, required : true },
    rating : {     : Number,     }
    comment : {    : String,     }
}, {
    timestamps : true
})
```

```
(cont'd...)

reviews : [reviewSchema],
rating : {
        type: Number,
        required : true,
        default : 0,
},
numReviews : {
        type : Number,
            "
  },
  price : {
        type : Number,
            "
    },
    countInStock : {
            "
    },
}, {
      timestamps : true
}
)

const Product = mongoose.model('Product', productSchema)

export default Product
```

orderModel.js

```
import mongoose from 'mongoose'

const orderSchema = mongoose.Schema(
  {
    user : {
      type : mongoose.Schema.Types.ObjectId,
      required : true,
      ref : 'User'
    },
    orderItems : [
      {
        name : { type : String, required : true },
        qty : {     : Number,       },
        image : {     : String,       },
        price : {     : Number,       },
        product : {
          type : mongoose.Schema.Types.ObjectId,
          required : true,
          ref : 'Product'
        }
      }
    ],
    shippingAddress : {
      address : { type : String, required : true },
      city :    {                                 },
```

```
    postalCode: { type: String, required: true },
    country: {                                  },
},
paymentMethod: {
    type: String,
    required: true
},
paymentResult: {
    id: { type: String },
    status: {          },
    update_time: {          },
    email_address: {          }
},
taxPrice: {
    type: Number,
    required: true,
    default: 0.0
},
shippingPrice: {
    (... copy taxPrice ... same)
},
totalPrice: {
    (same)
},
```

```
isPaid : {
        type : Boolean,
        required : true,
        default : false,
    },

    paidAt : {
        type : Date
    },

    isDelivered : {
        (... same as isPaid)
    },

    ~~paidAt~~ :

        deliveredAt : {
            type : Date
        },
    }, {
        timestamps : true,
    }
)

Const Order = mongoose.model('Order, orderSchema)

export default Order;
```