

Q-1 React Basics (JSX, Components, Props)

Product Card

```
ProductCard.jsx X
frontend > src > components > ProductCard.jsx > ...
1 import React from 'react'
2 import PropTypes from 'prop-types'
3 export default function ProductCard({ title, price, discount }){
4   const finalPrice = Number(price) - Number(discount);
5   return (
6     <div style={{ border: '1px solid #ddd', padding: 12, width: 260 }}>
7       <h3>{title}</h3>
8       <p>Price: ₹{price}</p>
9       <p>Discount: ₹{discount}</p>
10      <p><strong>Final Price: ₹{finalPrice}</strong></p>
11      <button>Shop Now</button>
12    </div>
13  );
14 }
15 ProductCard.propTypes = {
16   title: PropTypes.string.isRequired,
17   price: PropTypes.oneOfType([PropTypes.string, PropTypes.number]).isRequired,
18   discount: PropTypes.oneOfType([PropTypes.string, PropTypes.number]).isRequired
19 };
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

node - frontend + - - - | ↺ ×

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled **successfully**

Ln 1, Col 1 Spaces: 2 UTF-8 LF JavaScript JSX Go Live



ShopNow

[Home](#)

Sneakers

Price: ₹120

Discount: ₹20

Final Price: ₹100

Shop Now

Q-2 React State + Controlled and Uncontrolled Components

```
LoginForm.jsx X
frontend > src > components > LoginForm.jsx > LoginForm
1 import React, { useState, useRef } from 'react'
2 export default function LoginForm(){
3   const [username, setUsername] = useState('');
4   const passwordRef = useRef();
5   function handleSubmit(e){
6     e.preventDefault();
7     console.log('username:', username);
8     console.log('password:', passwordRef.current.value);
9     alert('Logged (check console)');
10  }
11  return (
12    <form onSubmit={handleSubmit} style={{ maxWidth: 360 }}>
13      <div>
14        <label>Username</label>
15        <input value={username} onChange={e => setUsername(e.target.value)} />
16      </div>
17      <div>
18        <label>Password</label>
19        <input type="password" ref={passwordRef} />
20      </div>
21      <button type="submit">Submit</button>
22    </form>
23  );
24 }
```




ShopNow

[Home](#) [Login](#)

Username

Password

Q-3 React Class Component, Lifecycle, PropTypes, Styling



The screenshot shows a VS Code editor with a file named `UserStatus.jsx` open. The code is as follows:

```

1 import React from 'react'
2 import PropTypes from 'prop-types'
3 export default class UserStatus extends React.Component{
4   constructor(props){
5     super(props);
6     this.state={message:'Fetching user status...'};
7   }
8   componentDidMount(){
9     this.timer=setTimeout(()=>{this.setState({message:'Active User'})},2000);
10  }
11  componentWillUnmount(){ clearTimeout(this.timer); }
12  render(){
13    return <div style={{padding:8, background:'#f6f6f6'}}>{this.state.message}</div>;
14  }
15 }
16 UserStatus.propTypes={ userId:PropTypes.number.isRequired };

```

Below the editor, the **TERMINAL** tab is active, showing the following output:

```

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully

```



ShopNow

[Home](#) | [Login](#)

Sneakers

Price: ₹120

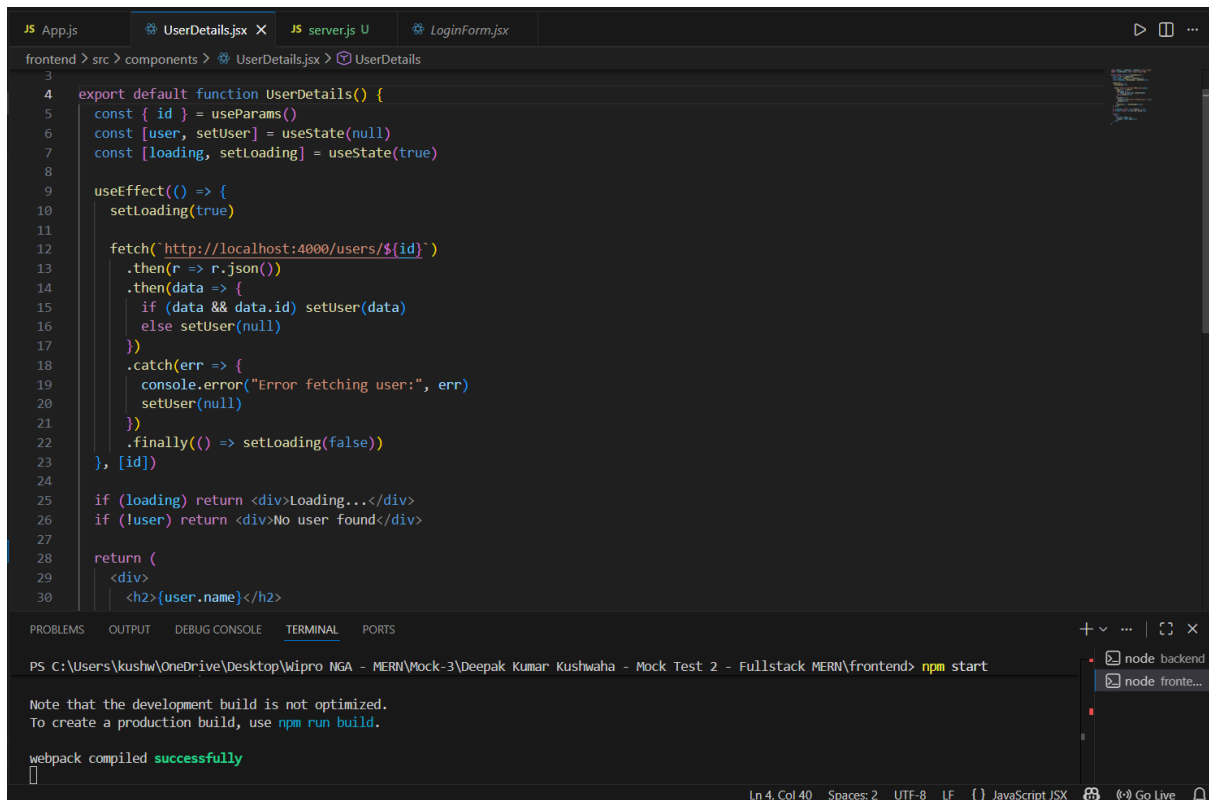
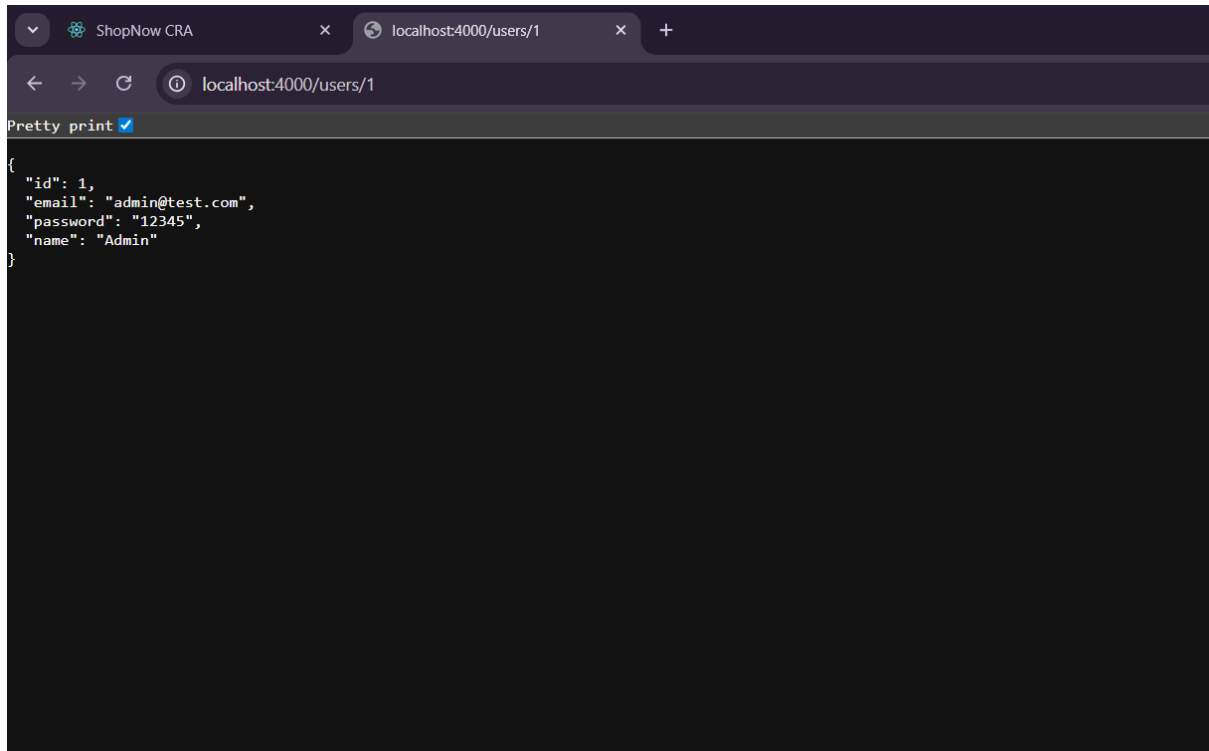
Discount: ₹20

Final Price: ₹100

[Shop Now](#)

Active User

Q-4 React Router + API Integration





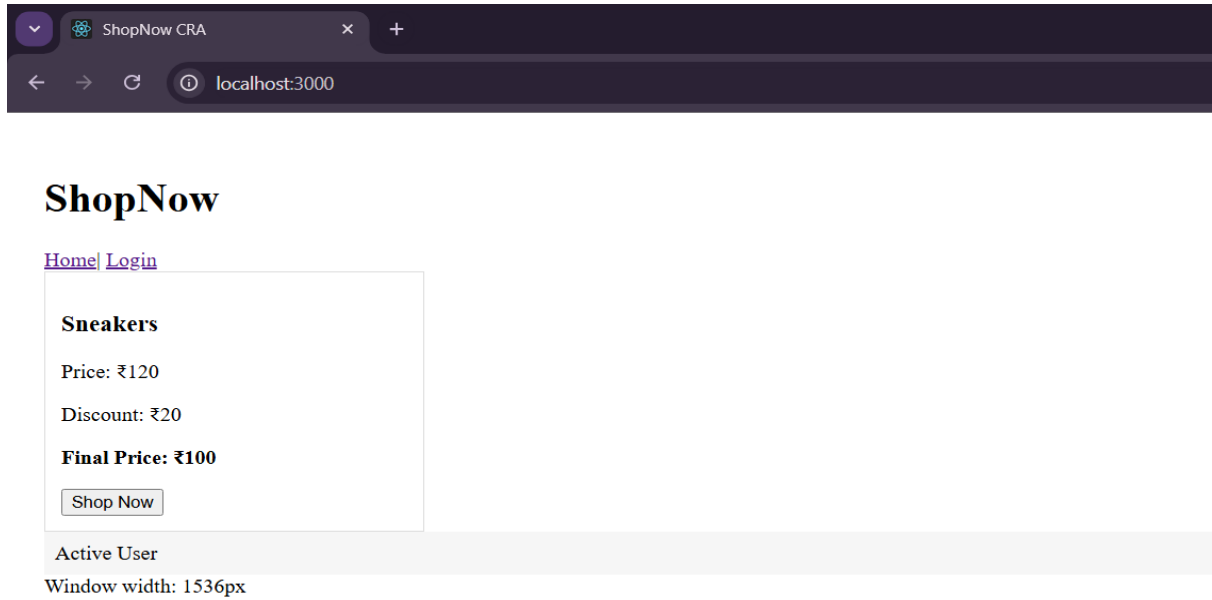
ShopNow

[Home](#) | [Login](#)

Admin

Email: admin@test.com

Q-5 Reusability Using HOC or Render Props





ShopNow

[Home](#) | [Login](#)

Sneakers

Price: ₹120

Discount: ₹20

Final Price: ₹100

Shop Now

Active User

Window width: 842px

Q-6 Formik + Yup Validation

```
JS App.js  FormikLogin.jsx  UserDetails.jsx  JS server.js U
frontend > src > components > FormikLogin.jsx > ...
1 import React from 'react'
2 import { Formik, Field, Form, ErrorMessage } from 'formik'
3 import * as Yup from 'yup'
4 const schema=Yup.object().shape({
5   email:Yup.string().email().required(),
6   password:Yup.string().min(6).required()
7 });
8 export default function FormikLogin(){
9   return (
10     <Formik initialValues={{email:'',password:''}} validationSchema={schema}
11       onSubmit={v=>{console.log(v);alert('Submitted')}}>
12       <Form>
13         <div><label>Email</label><Field name="email" /><ErrorMessage name="email" /></div>
14         <div><label>Password</label><Field name="password" type="password" /><ErrorMessage name="password" /></div>
15         <button type="submit">Login</button>
16       </Form>
17     </Formik>
18   );
19 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\frontend> npm start
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```



ShopNow

[Home](#) | [Login](#) | [Formik](#)

Email email must be a valid email
Password password is a required field

Q-7 Node.js Core Modules

The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'backend' folder containing 'cli.js' and 'server.js'. The code editor shows the content of 'cli.js', which is a Node.js CLI application. The application uses the 'fs' module to create a log directory and file, and the 'http' module to create a simple server. The terminal at the bottom shows the command 'node cli.js' being executed, which results in 'CLI server 5000' being printed to the console.

```
backend > JS cli.js > ...
1  const fs=require('fs')
2  const path=require('path')
3  const http=require('http')
4
5  const logDir=path.join(__dirname,'logs')
6  if(!fs.existsSync(logDir)) fs.mkdirSync(logDir)
7  const file=path.join(logDir,'app.log')
8  fs.appendFileSync(file,'App started\n')
9
10 http.createServer((req,res)=>{
11   res.setHeader('Content-Type','application/json')
12   res.end(JSON.stringify({status:'running'}))
13 }).listen(5000,()=>console.log("CLI server 5000"))
14
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

To address all issues, run:

```
GET /users/1
GET /users/1
GET /users/1
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node cli.js
CLI server 5000
```

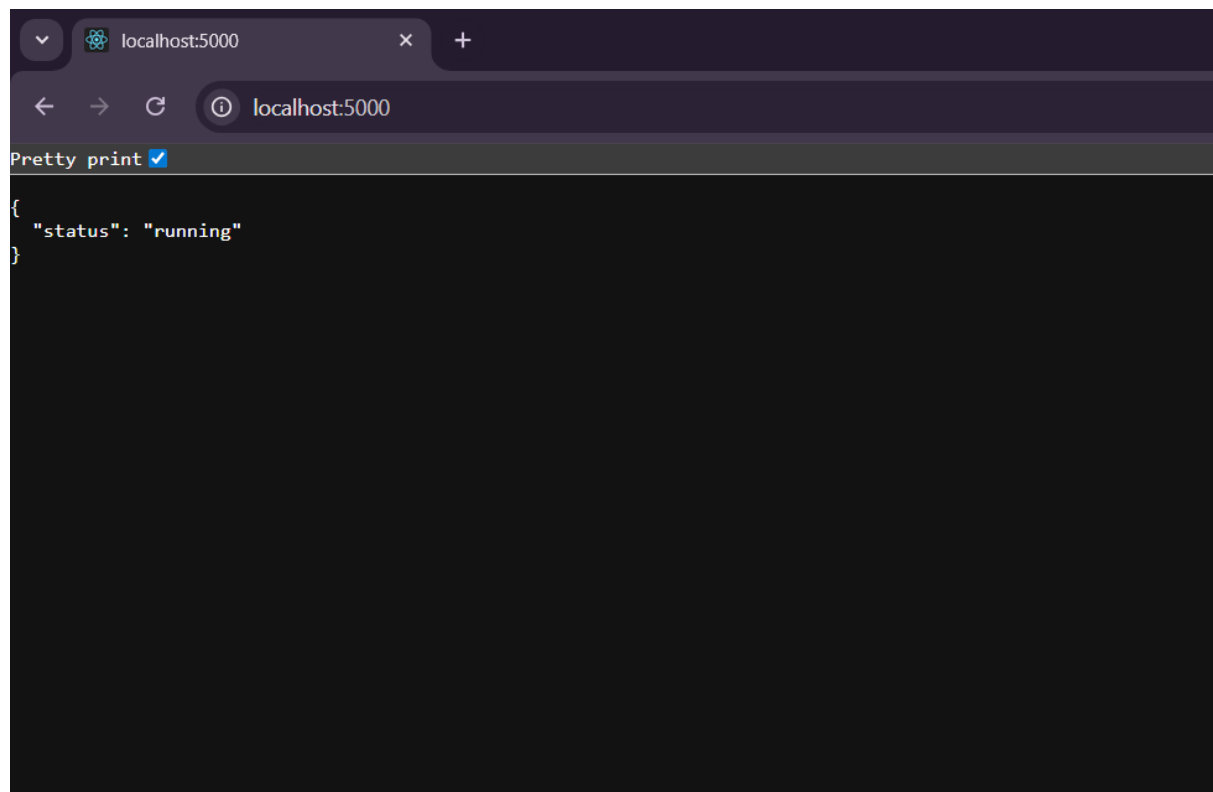
The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with a 'backend' folder containing 'cli.js' and 'server.js'. The code editor shows the content of 'cli.js', which is a Node.js CLI application. The terminal at the bottom shows the command 'node cli.js' being executed, which results in 'CLI server 5000' being printed to the console.

```
backend > logs > app.log
1  App started
2
```

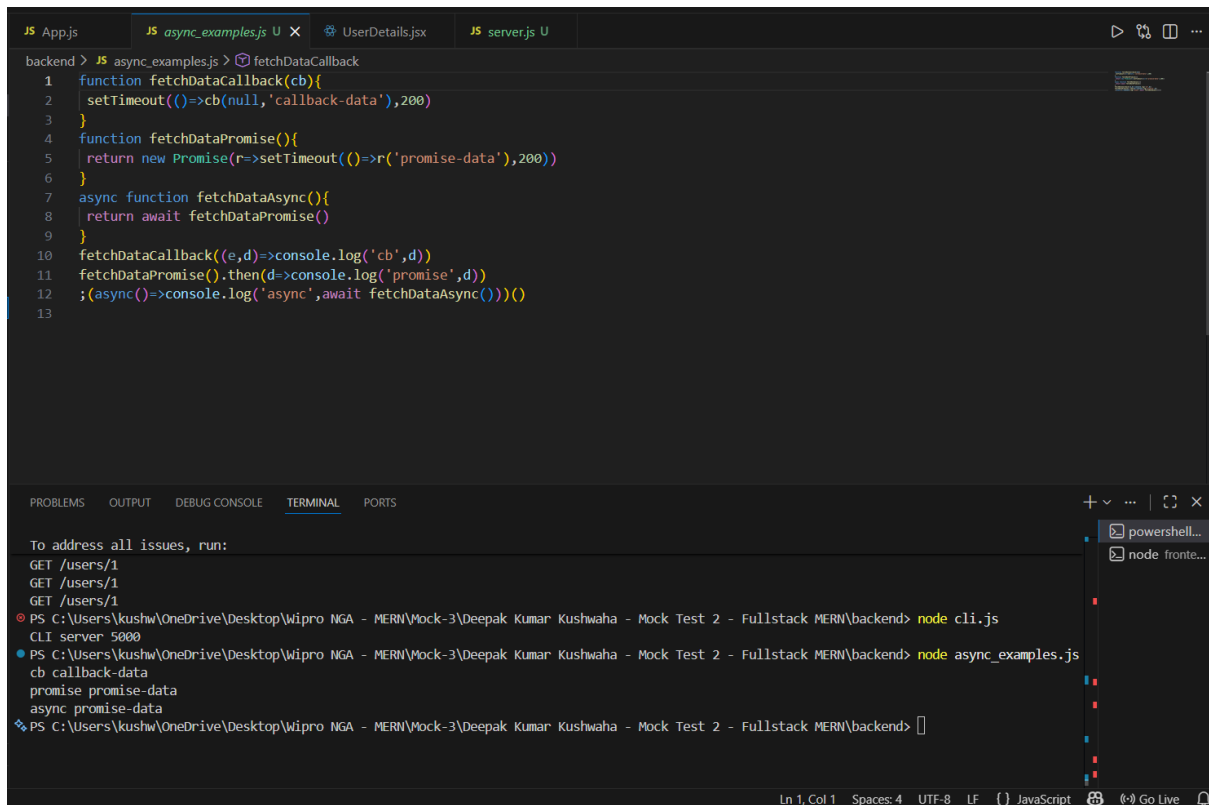
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

To address all issues, run:

```
GET /users/1
GET /users/1
GET /users/1
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node cli.js
CLI server 5000
```



Q-8 Asynchronous JavaScript (Callbacks → Promise → Async/Await)



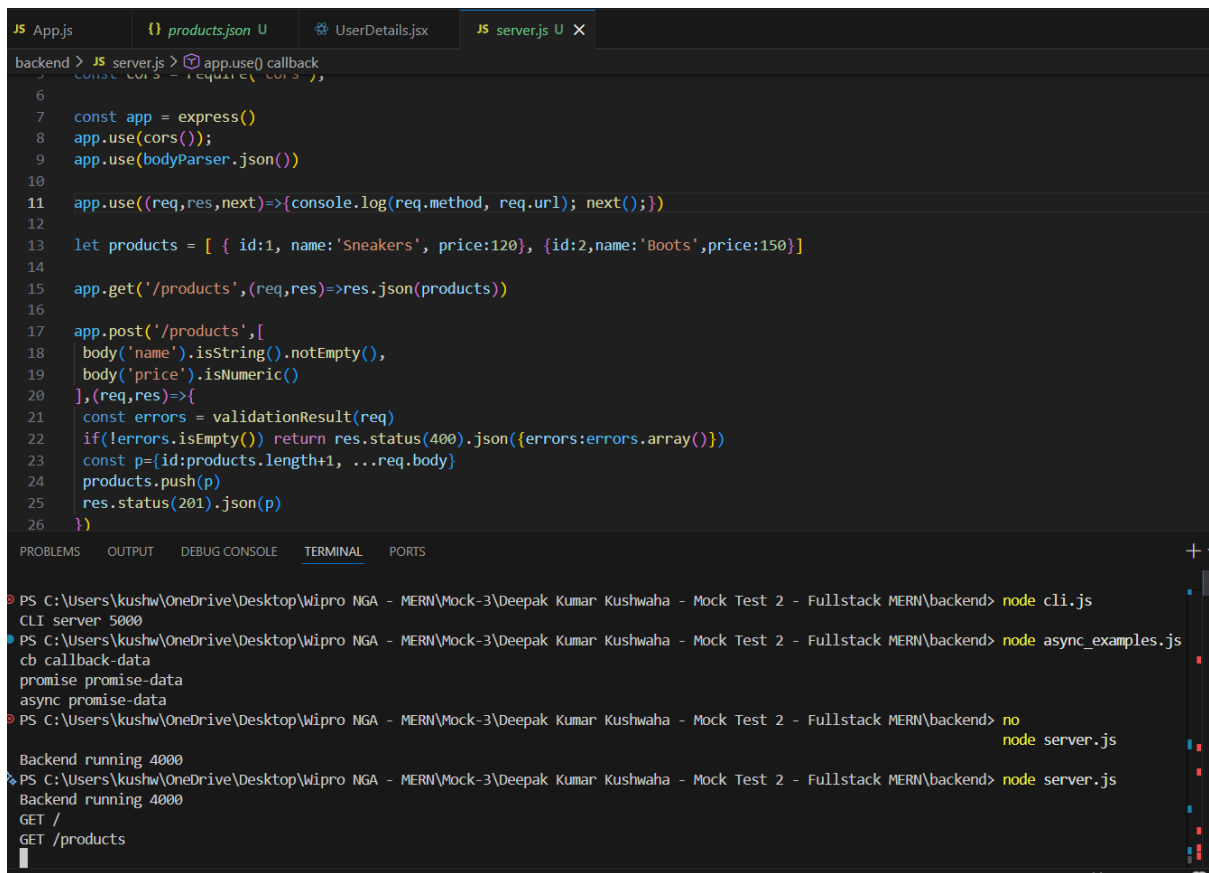
The image shows a VS Code editor window with a file named `async_examples.js` open. The code defines three functions: `fetchDataCallback` (using a callback), `fetchDataPromise` (using a Promise), and `fetchDataAsync` (using async/await). The `fetchDataAsync` function calls `fetchDataPromise` and logs the result. The terminal output shows the execution of `node cli.js` and `node async_examples.js`, resulting in the following logs:

```
GET /users/1
GET /users/1
GET /users/1
PS C:\Users\kushw\OneDrive\Desktop\Wipro_NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node cli.js
CLI server 5000
cb callback-data
promise promise-data
async promise-data
PS C:\Users\kushw\OneDrive\Desktop\Wipro_NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend>
```

The code in `async_examples.js` is as follows:

```
1 function fetchDataCallback(cb){
2   setTimeout(()=>cb(null,'callback-data'),200)
3 }
4 function fetchDataPromise(){
5   return new Promise(r=>setTimeout(()=>r('promise-data'),200))
6 }
7 async function fetchDataAsync(){
8   return await fetchDataPromise()
9 }
10 fetchDataCallback((e,d)=>console.log('cb',d))
11 fetchDataPromise().then(d=>console.log('promise',d))
12 ;(async()=>console.log('async',await fetchDataAsync()))()
13
```

Q-9 Express Routing + Middleware + Validation

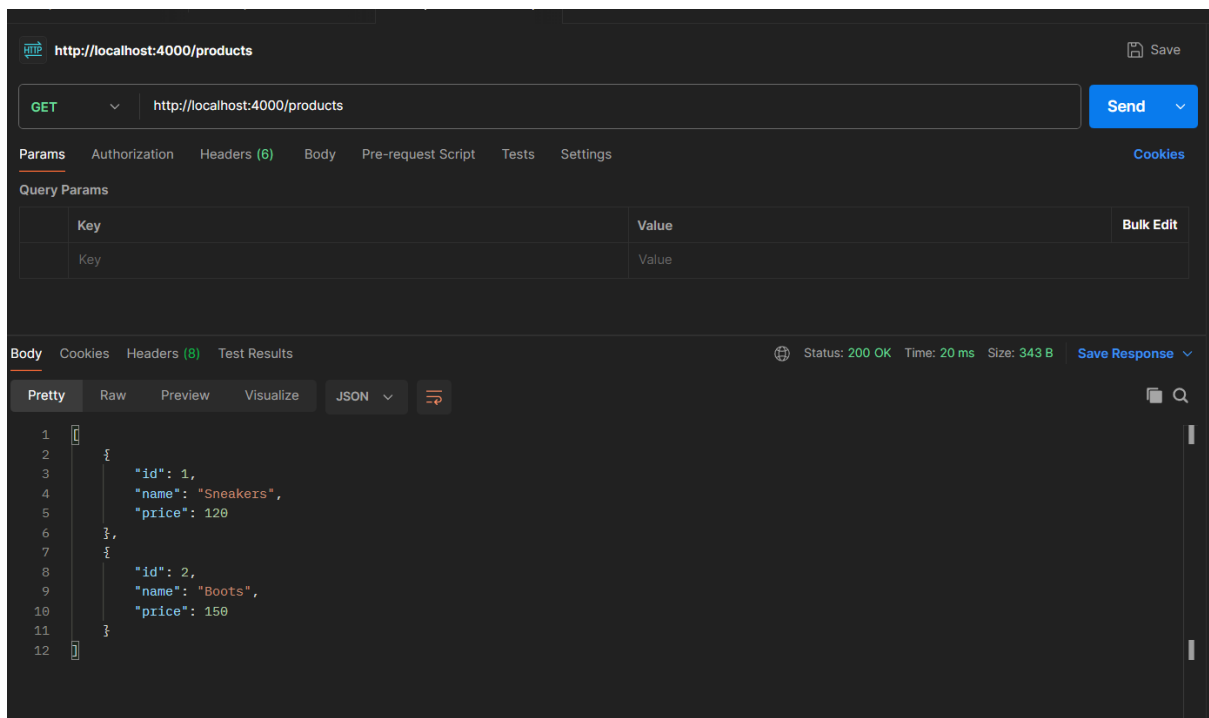


The image shows a VS Code editor with a file named `server.js` open. The code implements an Express.js server with the following features:

- Imports `express` and `bodyParser`.
- Creates an Express app and uses `bodyParser.json()` for JSON parsing.
- Defines a `products` array: `let products = [{ id:1, name:'Sneakers', price:120}, {id:2,name:'Boots',price:150}]`.
- Routes `GET /products` to return the `products` array as JSON.
- Routes `POST /products` with validation using `validationResult`. It checks if `body('name')` is a non-empty string and `body('price')` is a numeric value. If validation fails, it returns a 400 status with the error details. If successful, it pushes the new product to the `products` array and returns a 201 status with the new product.

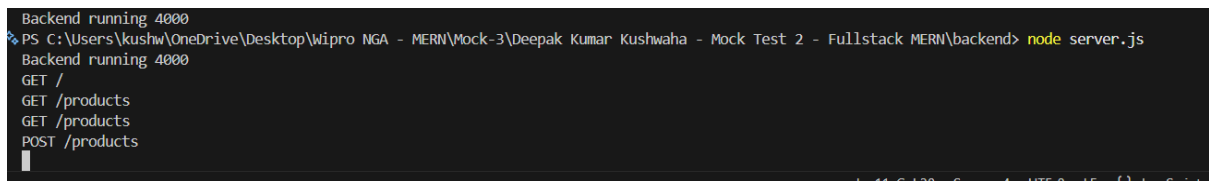
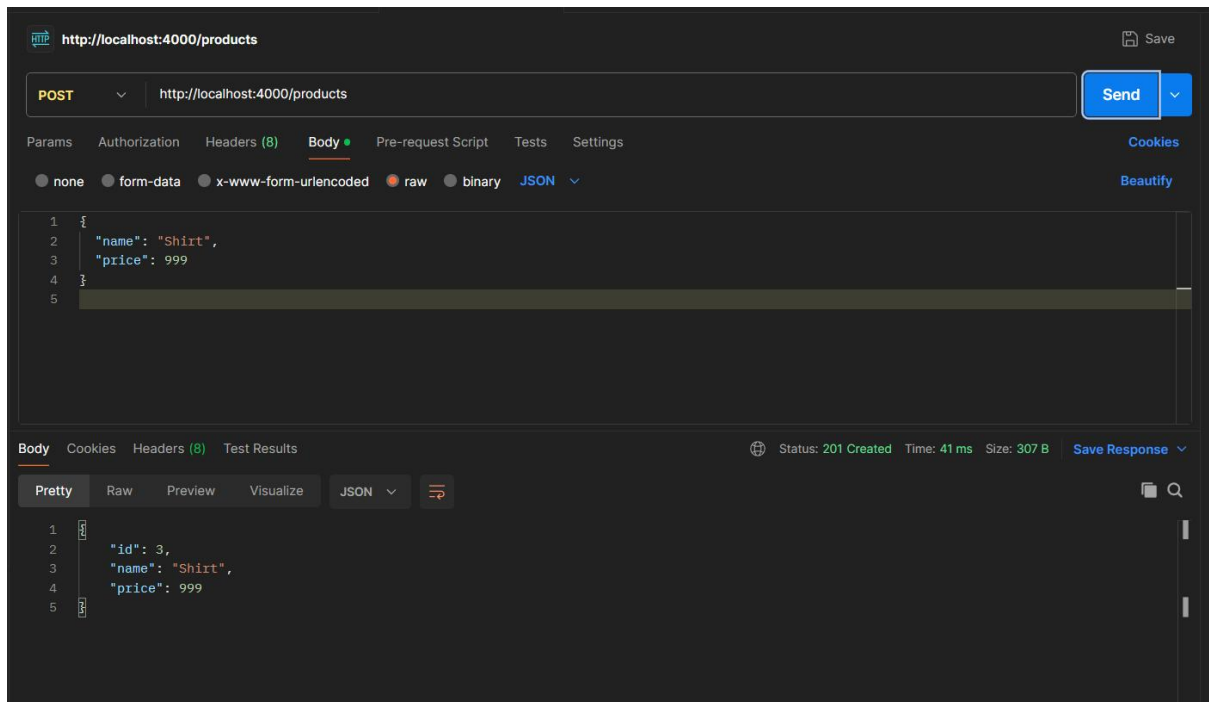
The terminal at the bottom shows the following commands and output:

```
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node cli.js
CLI server 5000
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node async_examples.js
cb callback-data
promise promise-data
async promise-data
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> no
node server.js
Backend running 4000
PS C:\Users\kushw\OneDrive\Desktop\Wipro NGA - MERN\Mock-3\Deepak Kumar Kushwaha - Mock Test 2 - Fullstack MERN\backend> node server.js
Backend running 4000
GET /
GET /products
```

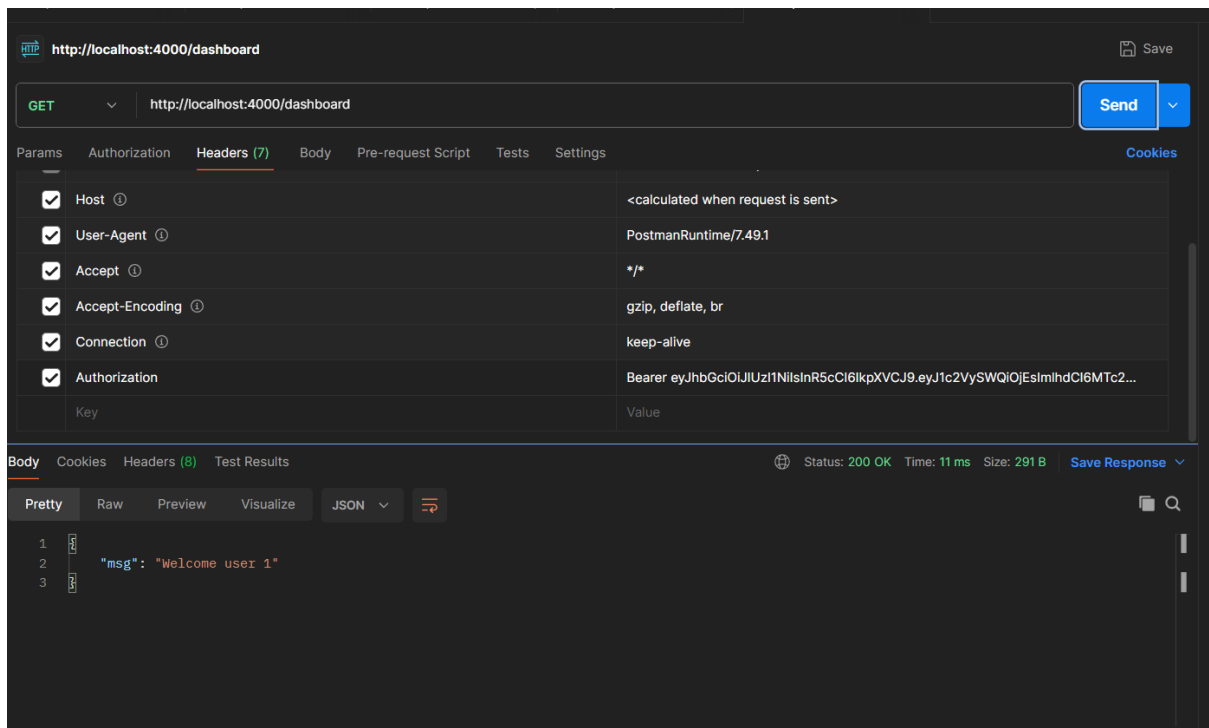
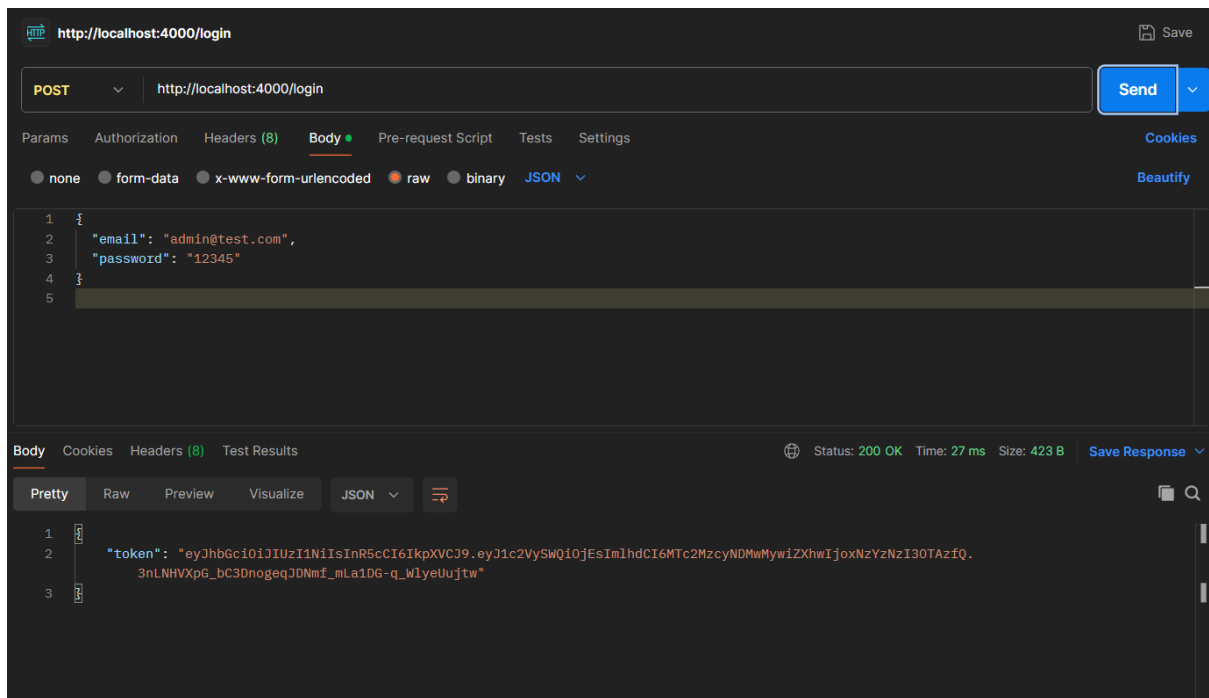


The image shows the Chrome DevTools Network tab with a request to `http://localhost:4000/products` selected. The request method is `GET`. The response status is `200 OK`, with a time of `20 ms` and a size of `343 B`. The response body is displayed in JSON format:

```
{
  "id": 1,
  "name": "Sneakers",
  "price": 120
},
{
  "id": 2,
  "name": "Boots",
  "price": 150
}
```



Q-10 REST API + JWT Authentication



HTTP

http://localhost:4000/dashboard

Save

GET

http://localhost:4000/dashboard

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

☒

Host ⓘ

<calculated when request is sent>

☒

User-Agent ⓘ

PostmanRuntime/7.49.1

☒

Accept ⓘ

/

☒

Accept-Encoding ⓘ

gzip, deflate, br

☒

Connection ⓘ

keep-alive

☒

Authorization

Bearer

KeyValue

Body

Cookies

Headers (8)

Test Results

Status: 401 Unauthorized

Time: 7 ms

Size: 302 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

"error": "Invalid token"