

# Mongoose

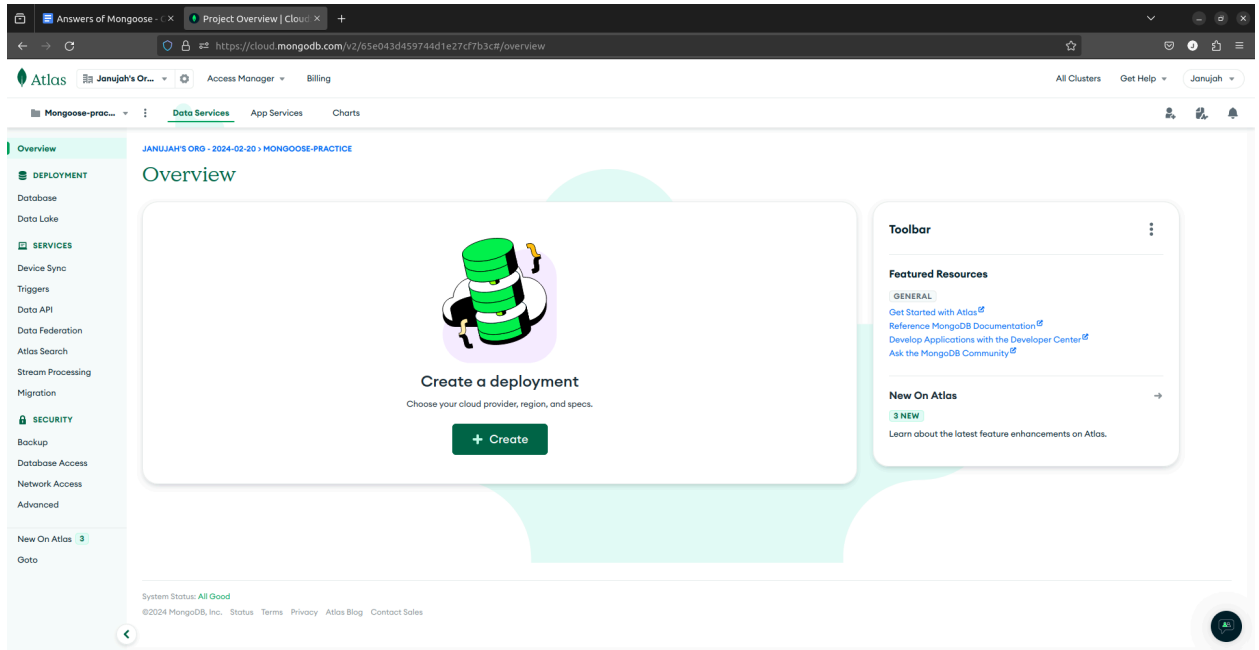
## Foundation Questions

1. **Mongoose** is an Object Data Modeling (ODM) library built for MongoDB and JavaScript. It's to help with data modeling, schema, model validation, and general data manipulation in MongoDB.
2. Database Interaction, Data Modeling and Validation, Querying and Poppulation, Middleware and hooks, Abstraction and Productivity.
3. **Mongoose schemas** are how to tell Mongoose what your documents look like. It is important to define the structure of our document and casting of properties, and also define the document instance of methods, static model methods, compound indexes, and document lifecycle hooks called middleware.
4. Models are modified constructors compiled from schema definitions. An instance of a model is called a document. Models are responsible for creating and reading documents from the underlying MongoDB database.
5. **save()** can be used to both create a new record and update an existing record. **create()** is used to create a new record by providing all required field at one time.
6. **CRUD operations(Create, Read, Update, Delete)**  
**Create:** In MongoDB, you can create a new document using the insertOne() or insertMany() methods.  
**Read:** You can read or retrieve documents from a MongoDB database using the find() method.  
**Update:** Updating documents in MongoDB can be done using the updateOne(), updateMany(), or replaceOne() methods.  
**Delete:** You can delete documents using the deleteOne() or deleteMany() methods.
7. **Middleware**, also known as pre and post-hooks, are functions that intercept the execution of asynchronous operations in Mongoose. It acts as a bridge between diverse technologies, tools, and databases
8. mongoose.connect(insert your MongoDB Atlas URI)
- 9.

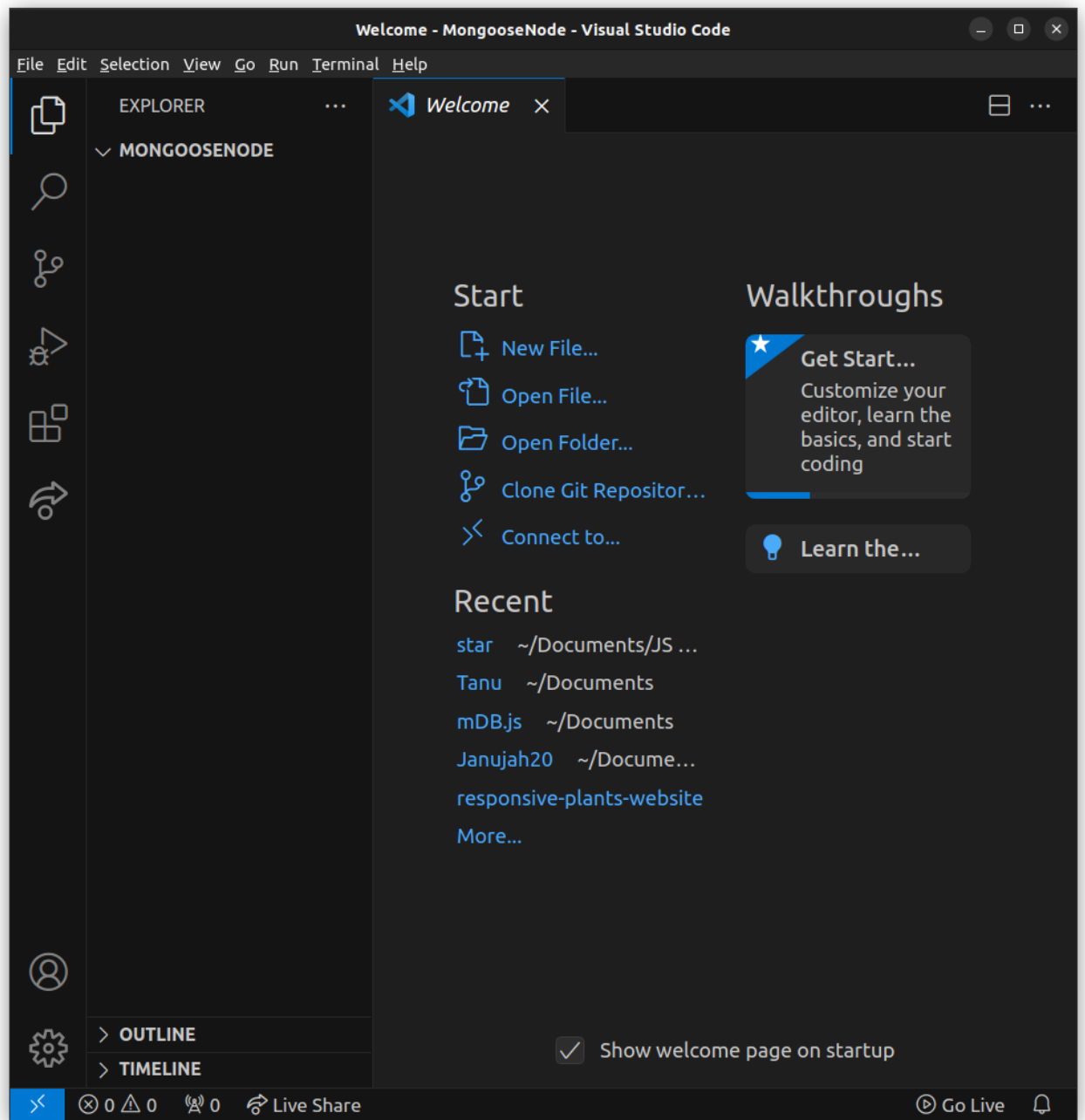
## Deep Dive into the Concepts

1. Async is multi-thread, which means operations or programs can run in parallel. Sync is a single thread, so only one operation or program will run at a time

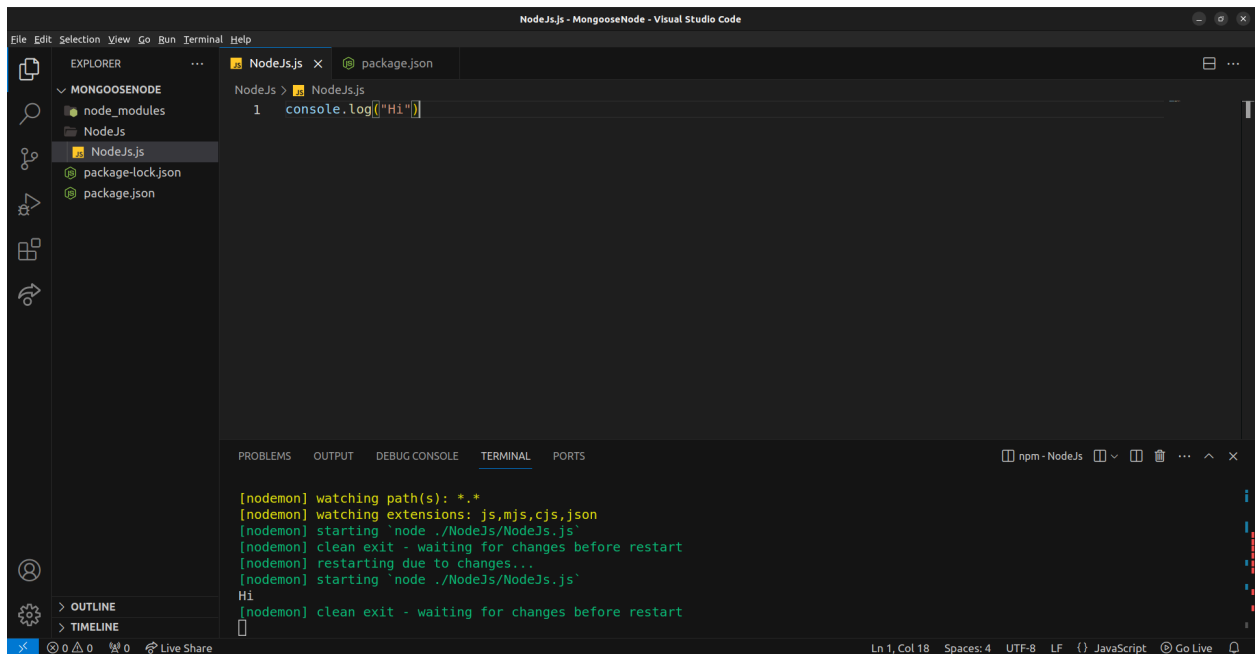
2.



3.



4.

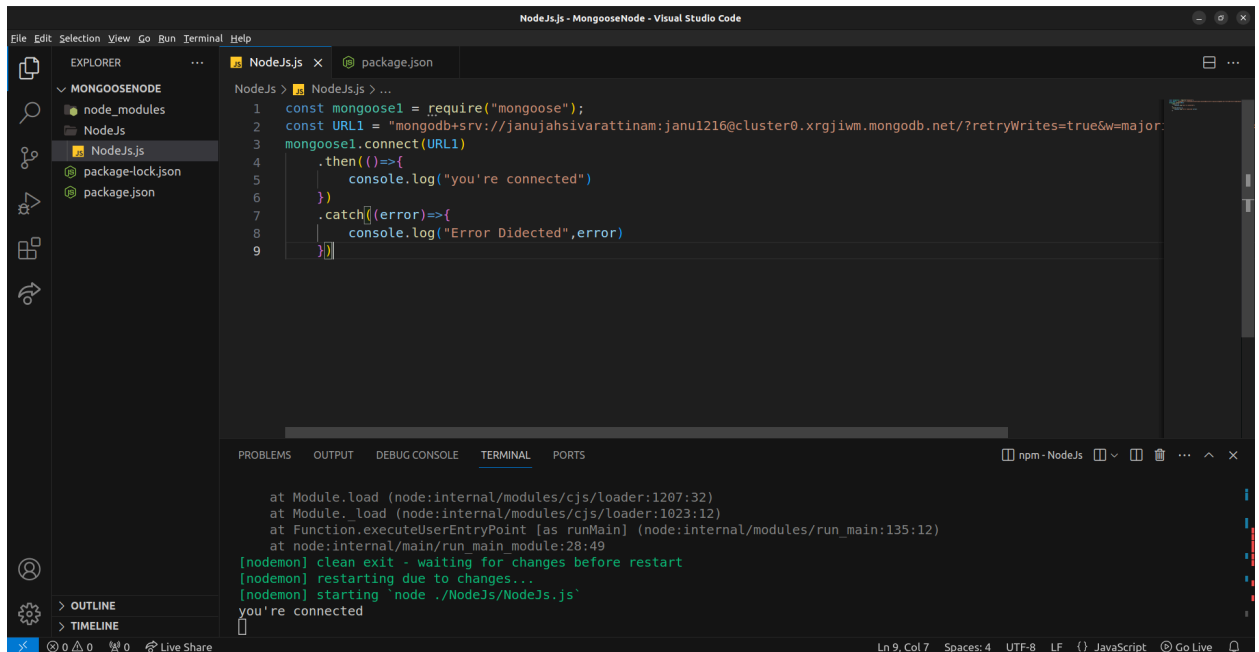


```
NodeJs.js > NodeJs.js
1 console.log('Hi');
```

```
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./NodeJs/NodeJs.js`
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node ./NodeJs/NodeJs.js`
Hi
[nodemon] clean exit - waiting for changes before restart
```

5. **Nodemon** is a command line tool that helps develop Node.js based applications by automatically restarting the code application when file changes in the directory are detected.

6.

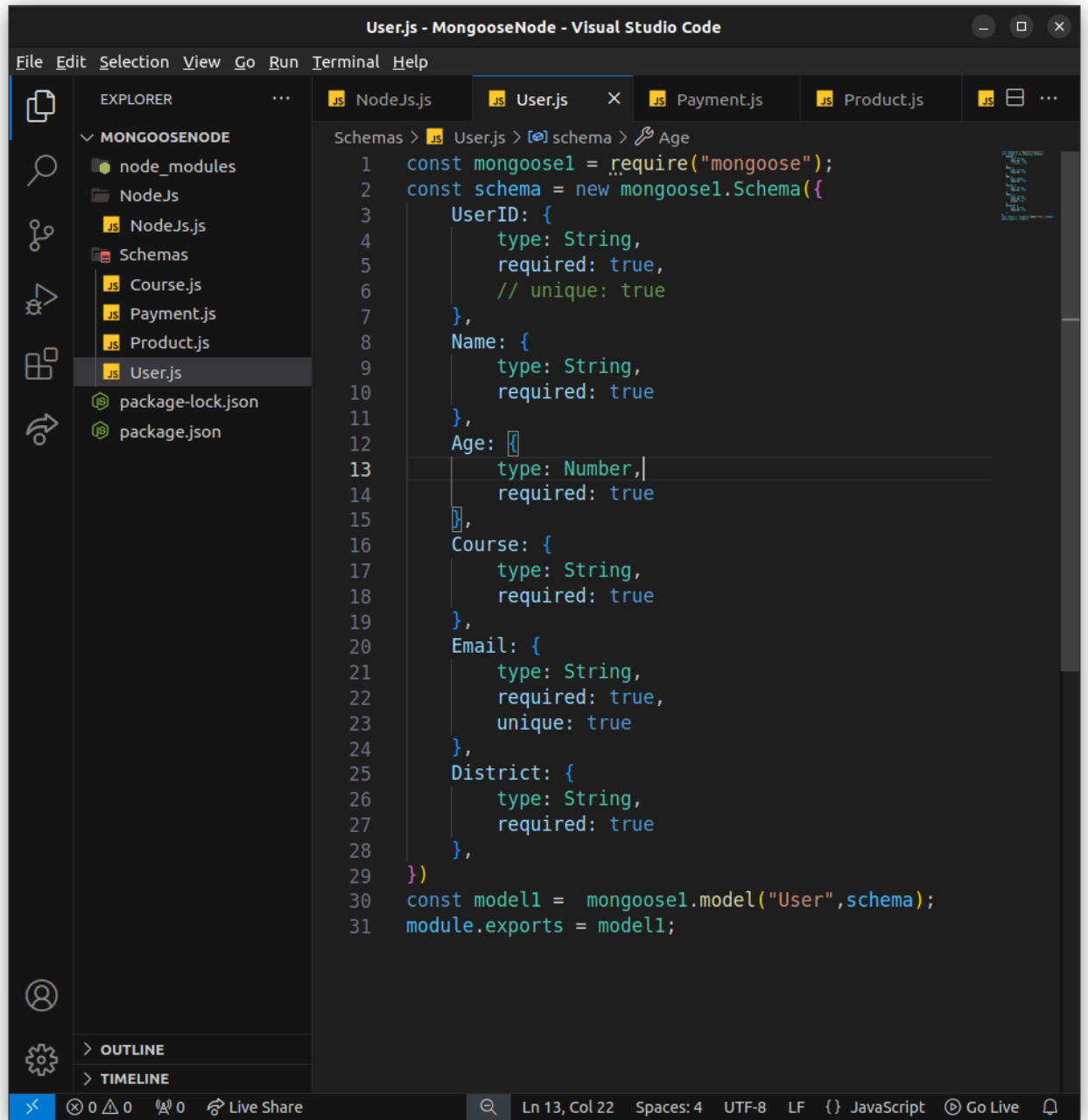


```
NodeJs.js > ...
1 const mongoose1 = require("mongoose");
2 const URL1 = "mongodb+srv://janujahsivarattinam:janul216@cluster0.xrgjiwm.mongodb.net/?retryWrites=true&w=majority";
3 mongoose1.connect(URL1)
4   .then(()=>{
5     console.log("you're connected")
6   })
7   .catch((error)=>{
8     console.log("Error Didedcted",error)
9   })
```

```
at Module.load (node:internal/modules/cjs/loader:1207:32)
at Module._load (node:internal/modules/cjs/loader:1023:12)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:135:12)
at node:internal/main/run_main module:28:49
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node ./NodeJs/NodeJs.js`
you're connected
```

7.

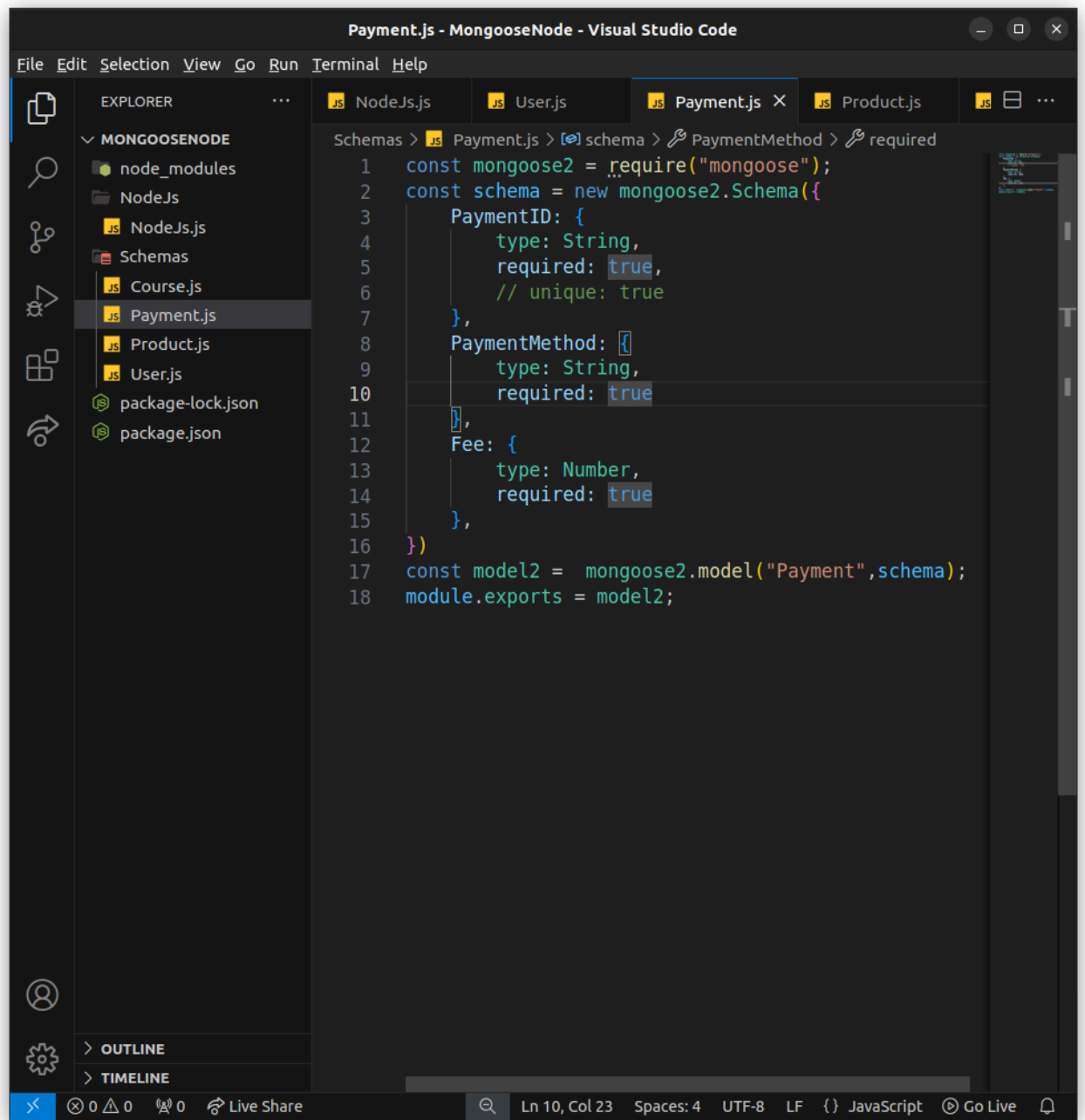
a. User's Schema



The screenshot shows the Visual Studio Code editor with the file 'User.js' open. The Explorer sidebar on the left shows a project structure for 'MongooseNode' with files like 'Node.js', 'Schemas', 'Course.js', 'Payment.js', 'Product.js', 'User.js', 'package-lock.json', and 'package.json'. The 'User.js' file is selected, and its content is displayed in the main editor. The code defines a Mongoose schema for a 'User' model with fields: 'UserID' (String, required, unique), 'Name' (String, required), 'Age' (Number, required), 'Course' (String, required), 'Email' (String, required, unique), and 'District' (String, required). The schema is then used to create a model and exported.

```
1 const mongoose = require("mongoose");
2 const schema = new mongoose.Schema({
3   UserID: {
4     type: String,
5     required: true,
6     // unique: true
7   },
8   Name: {
9     type: String,
10    required: true
11  },
12  Age: {
13    type: Number,
14    required: true
15  },
16  Course: {
17    type: String,
18    required: true
19  },
20  Email: {
21    type: String,
22    required: true,
23    unique: true
24  },
25  District: {
26    type: String,
27    required: true
28  },
29 })
30 const model = mongoose.model("User", schema);
31 module.exports = model;
```

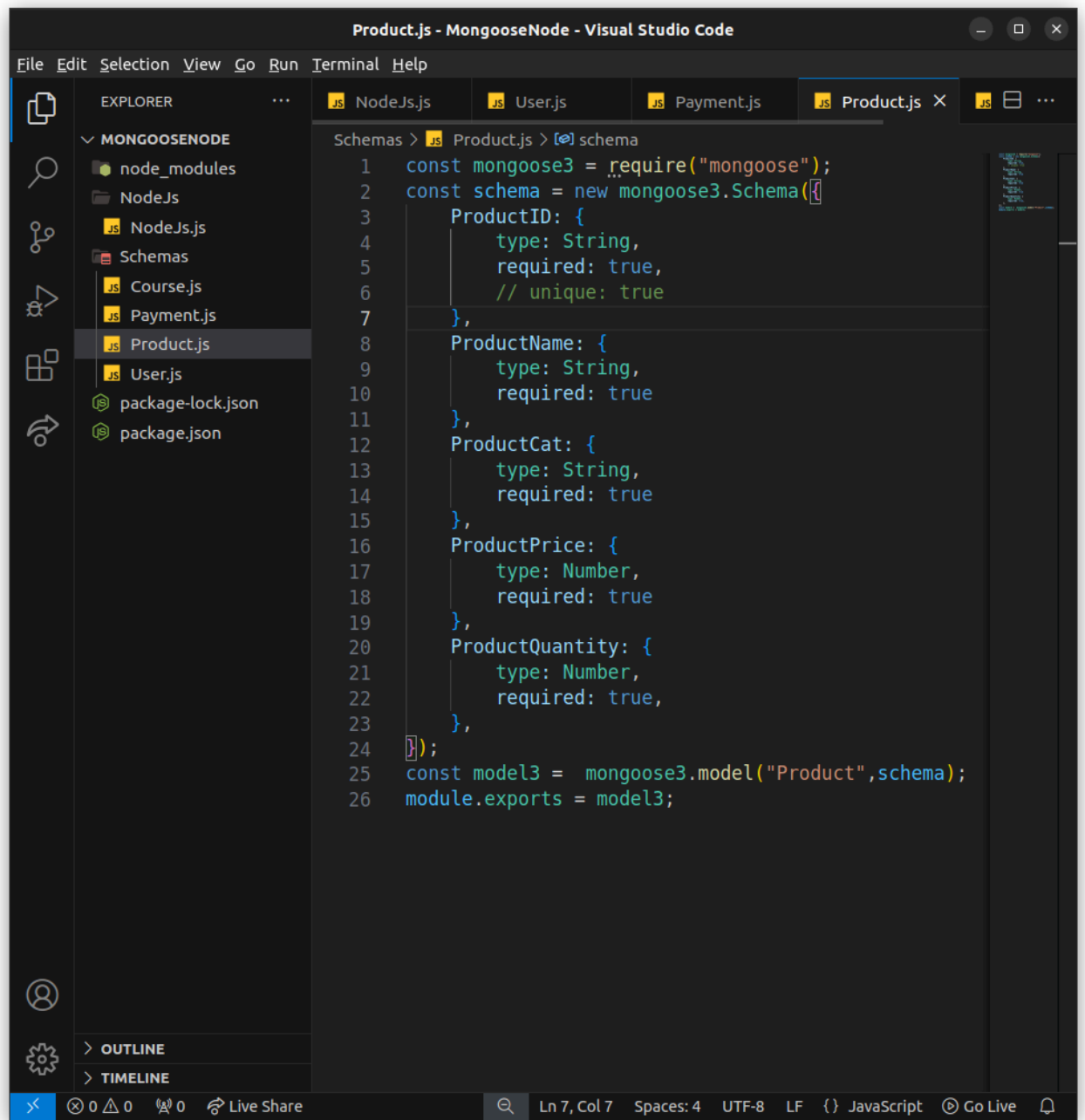
## b. Payment's Schema



The screenshot shows the Visual Studio Code editor with the file `Payment.js` open. The Explorer sidebar on the left shows a project named `MONGOOSENODE` with files `node_modules`, `Node.js`, `NodeJs.js`, `Schemas`, `Course.js`, `Payment.js` (selected), `Product.js`, `User.js`, `package-lock.json`, and `package.json`. The bottom status bar indicates the file is `Payment.js` at line 10, column 23, with 4 spaces, UTF-8 encoding, and LF line endings.

```
1  const mongoose2 = require("mongoose");
2  const schema = new mongoose2.Schema({
3    PaymentID: {
4      type: String,
5      required: true,
6      // unique: true
7    },
8    PaymentMethod: {
9      type: String,
10     required: true
11   },
12   Fee: {
13     type: Number,
14     required: true
15   },
16 })
17 const model2 = mongoose2.model("Payment", schema);
18 module.exports = model2;
```

### c. Product's Schema

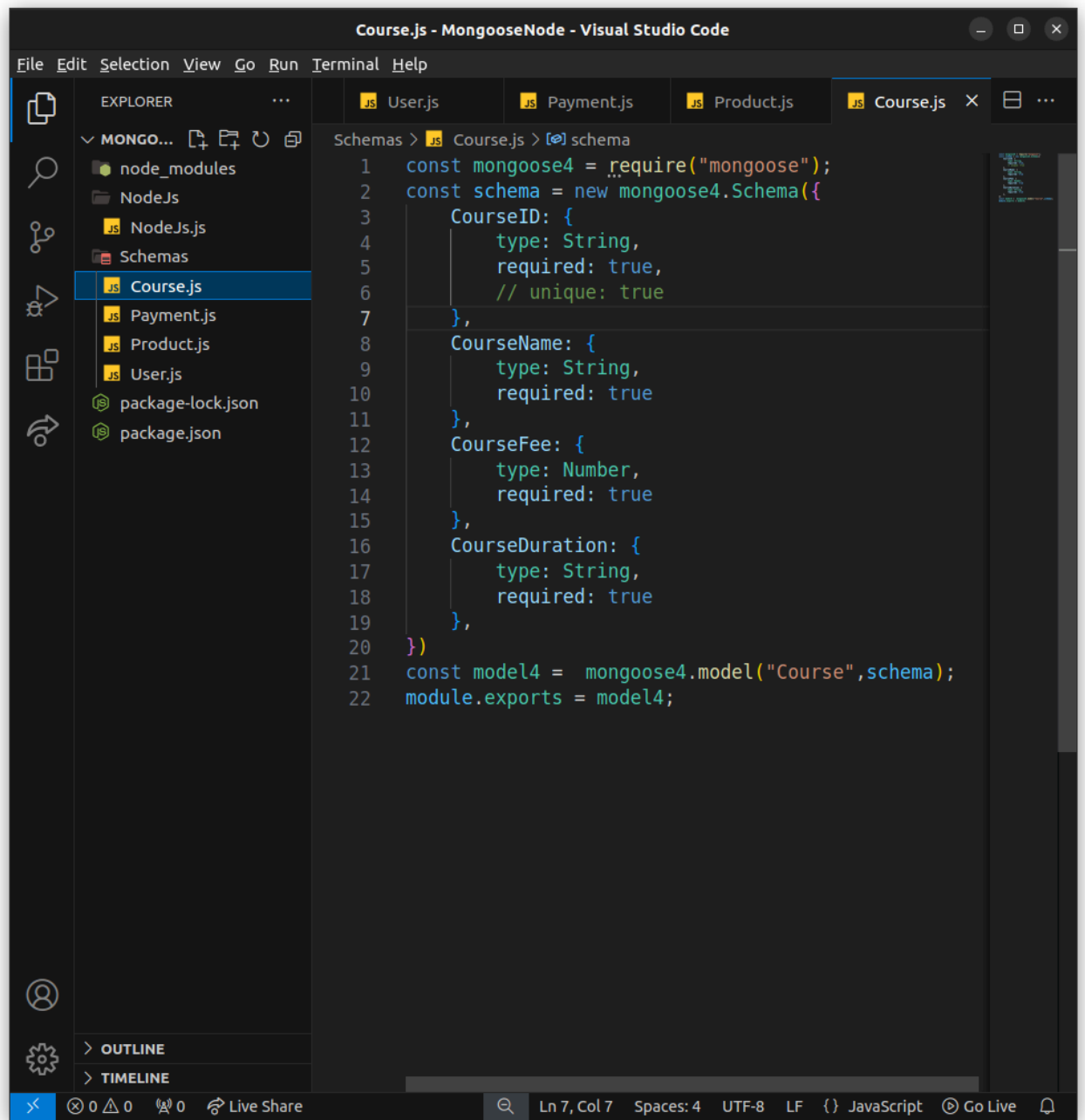


The screenshot shows the Visual Studio Code editor with the file `Product.js` open. The Explorer sidebar on the left shows the project structure for `MONGOOSENODE`, including `node_modules`, `Node.js`, `NodeJs.js`, `Schemas`, `Course.js`, `Payment.js`, `Product.js` (selected), `User.js`, `package-lock.json`, and `package.json`. The Editor pane displays the following JavaScript code:

```
1  const mongoose3 = require("mongoose");
2  const schema = new mongoose3.Schema({
3      ProductID: {
4          type: String,
5          required: true,
6          // unique: true
7      },
8      ProductName: {
9          type: String,
10         required: true
11     },
12     ProductCat: {
13         type: String,
14         required: true
15     },
16     ProductPrice: {
17         type: Number,
18         required: true
19     },
20     ProductQuantity: {
21         type: Number,
22         required: true,
23     },
24 });
25 const model3 = mongoose3.model("Product", schema);
26 module.exports = model3;
```

The status bar at the bottom indicates the current position is `Ln 7, Col 7`, with `Spaces: 4`, `UTF-8` encoding, `LF` line endings, and the file is a `JavaScript` document.

#### d. Course's Schema



The screenshot shows the Visual Studio Code editor with the 'Course.js' file open. The Explorer sidebar on the left shows a project structure with a 'MONGO...' folder containing 'node\_modules', 'Node.js', 'NodeJs.js', 'Schemas', 'Course.js', 'Payment.js', 'Product.js', 'User.js', 'package-lock.json', and 'package.json'. The 'Course.js' file is selected. The main editor area shows the following JavaScript code:

```
1  const mongoose4 = require("mongoose");
2  const schema = new mongoose4.Schema({
3    CourseID: {
4      type: String,
5      required: true,
6      // unique: true
7    },
8    CourseName: {
9      type: String,
10     required: true
11   },
12   CourseFee: {
13     type: Number,
14     required: true
15   },
16   CourseDuration: {
17     type: String,
18     required: true
19   },
20 })
21 const model4 = mongoose4.model("Course", schema);
22 module.exports = model4;
```

The status bar at the bottom indicates the cursor is at line 7, column 7, with 4 spaces, UTF-8 encoding, LF line endings, and the file is a JavaScript file.



## 8. Users

The screenshot shows the MongoDB Atlas web interface. The left sidebar contains navigation options: Overview, DEPLOYMENT, Database, Data Lake, SERVICES, Device Sync, Triggers, Data API, Data Federation, Atlas Search, Stream Processing, Migration, SECURITY, Backup, Database Access, Network Access, Advanced, and Oato. The 'Database' section is expanded, showing a search bar and a list of namespaces: test, courses, payments, products, and users. The 'users' namespace is selected. The main panel displays the 'test.users' collection with a storage size of 36KB, logical data size of 2.08KB, and 13 total documents. A 'Find' tab is active, showing a query filter bar with the text 'Type a query: { field: 'value' }'. Below the filter, the query results are displayed as a list of 13 documents. The first document is shown in detail, with fields: \_id, UserID, Name, Age, Courses, Email, and District. The system status at the bottom is 'All Good'.

test.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 2.08KB TOTAL DOCUMENTS: 13 INDEXES TOTAL SIZE: 108KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 1-13 OF 13

```
{
  "_id": "ObjectId('65e0a150b0413c5c4d73c1')",
  "UserID": "121635AB",
  "Name": "Janujah",
  "Age": 21,
  "Courses": "FPG",
  "Email": "janujahsivarattinam@gmail.com",
  "District": "Vavuniya",
  "__v": 0
}
```

System Status: All Good

©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

## Payments

The screenshot shows the MongoDB Atlas web interface. The left sidebar contains navigation options: Overview, DEPLOYMENT, Database, Data Lake, SERVICES, Device Sync, Triggers, Data API, Data Federation, Atlas Search, Stream Processing, Migration, SECURITY, Backup, Database Access, Network Access, Advanced, and Oato. The 'Database' section is expanded, showing a search bar and a list of namespaces: test, courses, payments, products, and users. The 'payments' namespace is selected. The main panel displays the 'test.payments' collection with a storage size of 36KB, logical data size of 1.39KB, and 15 total documents. A 'Find' tab is active, showing a query filter bar with the text 'Type a query: { field: 'value' }'. Below the filter, the query results are displayed as a list of 15 documents. The first document is shown in detail, with fields: \_id, PaymentID, PaymentMethod, Fee, and \_\_v. The system status at the bottom is 'All Good'.

test.payments

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.39KB TOTAL DOCUMENTS: 15 INDEXES TOTAL SIZE: 72KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 1-15 OF 15

```
{
  "_id": "ObjectId('65e0a3adc517071759f1a4b')",
  "PaymentID": "1211",
  "PaymentMethod": "Cash",
  "Fee": 1000,
  "__v": 0
}
```

System Status: All Good

©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

# Products

AtlasJanujah's Or...Access ManagerBilling

Mongoose-proc...Data ServicesApp ServicesCharts

Overview

DEPLOYMENT

Database

Data Lake

SERVICES

Device Sync

Triggers

Data API

Data Federation

Atlas Search

Stream Processing

Migration

SECURITY

Backup

Database Access

Network Access

Advanced

Oato

+ Create Database

Search Namespaces

test

courses

payments

products

users

test.products

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 188KB TOTAL DOCUMENTS: 13 INDEXES TOTAL SIZE: 72KB

FindIndexesSchema Anti-PatternsAggregationSearch Indexes

INSERT DOCUMENT

FilterType a query: { field: 'value' }ResetApplyOptions

QUERY RESULTS: 1-13 OF 13

\_id: ObjectId('65e0a93d657668782d86747')

ProductID: "1313"

ProductName: "Cash"

ProductCat: "Class item"

ProductPrice: 2000

ProductQuantity: 9

\_\_v: 0

\_id: ObjectId('65e0ae18269c187f873f1764')

ProductID: "1353"

ProductName: "Cash"

ProductCat: "Class item"

ProductPrice: 2000

ProductQuantity: 9

\_\_v: 0

\_id: ObjectId('65e0ae6c99e9d5ff0a918e69')

ProductID: "13623"

ProductName: "Cash"

System Status: All Good  
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

# Courses

AtlasJanujah's Or...Access ManagerBilling

Mongoose-proc...Data ServicesApp ServicesCharts

Overview

DEPLOYMENT

Database

Data Lake

SERVICES

Device Sync

Triggers

Data API

Data Federation

Atlas Search

Stream Processing

Migration

SECURITY

Backup

Database Access

Network Access

Advanced

Oato

+ Create Database

Search Namespaces

test

courses

payments

products

users

test.courses

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 229B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 108KB

FindIndexesSchema Anti-PatternsAggregationSearch Indexes

INSERT DOCUMENT

FilterType a query: { field: 'value' }ResetApplyOptions

QUERY RESULTS: 1-2 OF 2

\_id: ObjectId('65e13958cf5219569c0a90e2')

CourseId: "111111"

CourseName: "FWD"

CourseFee: 20000

CourseDuration: "6 Months"

\_\_v: 0

\_id: ObjectId('65e131640185d906254830a')

CourseID: "jjgk"

CourseName: "FWD"

CourseFee: 20000

CourseDuration: "6 Months"

\_\_v: 0

System Status: All Good  
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

9.

```
Node.js.js - MongooseNode - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
MONGOOSENODE
node_modules
Node.js
Node.js.js
Schemas
Course.js
Payment.js
Product.js
User.js
package-lock.json
package.json

Node.js.js
60 //
61 //   CourseFee: 20000,
62 //   CourseDuration: "6 Months",
63 // })
64 // course.save()
65 // .then(() =>{
66 //   console.log("Course registered")
67 // })
68 // .catch((err) =>{
69 //   console.log("error",err)
70 // });
71
72 async function fetchAllProducts() {
73   try {
74     const products = await model3.find({});
75     console.log('Products:', products);
76   } catch {
77     console.error('Error fetching products');
78   }
79 }
80
81 fetchAllProducts();
82
83

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

},
{
  id: new ObjectId('65e137c707dee50d3a5e947e'),
  ProductID: '16243E05gAtrwersCF84367sdfs23J',
  ProductName: 'Cash',
  ProductCat: 'Glass item',
  ProductPrice: 2000,
  ProductQuantity: 9,
  __v: 0
}

npm
npm
Ln 76, Col 16 Spaces: 4 UTF-8 LF () JavaScript Go Live
```