Mongoose

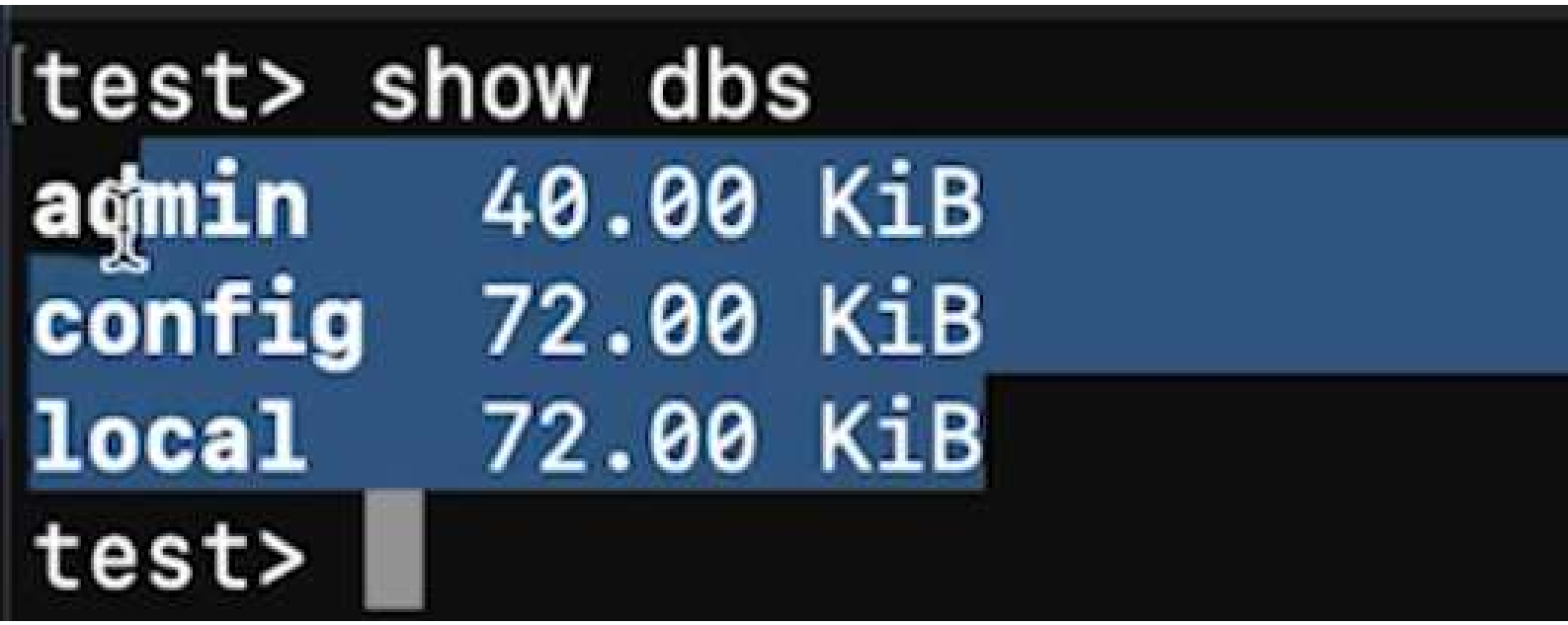
A library that creates a connection between MongoDB & Node.js JavaScript Runtime Environment

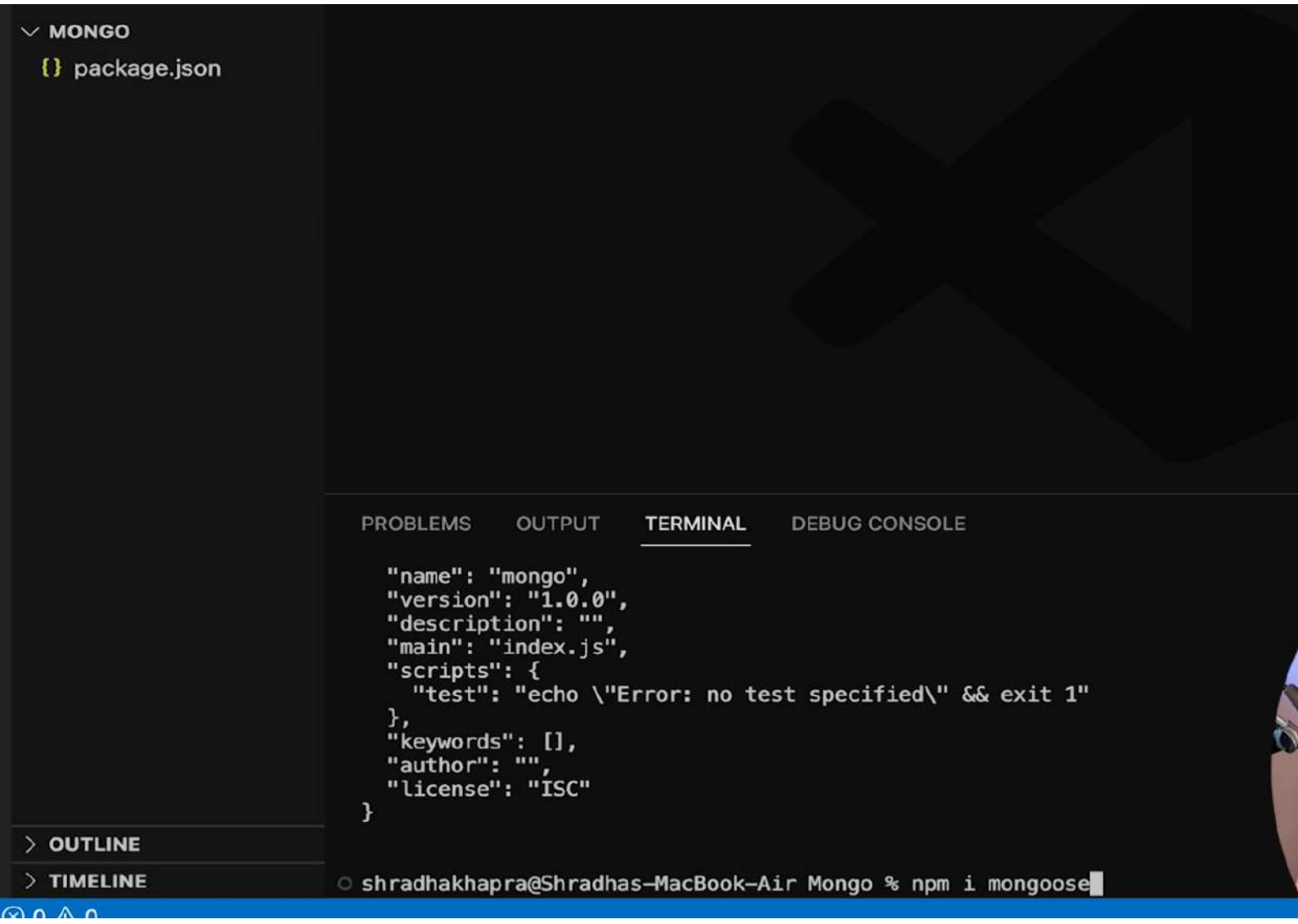
It is an ODM (Object Data Modeling) Library.

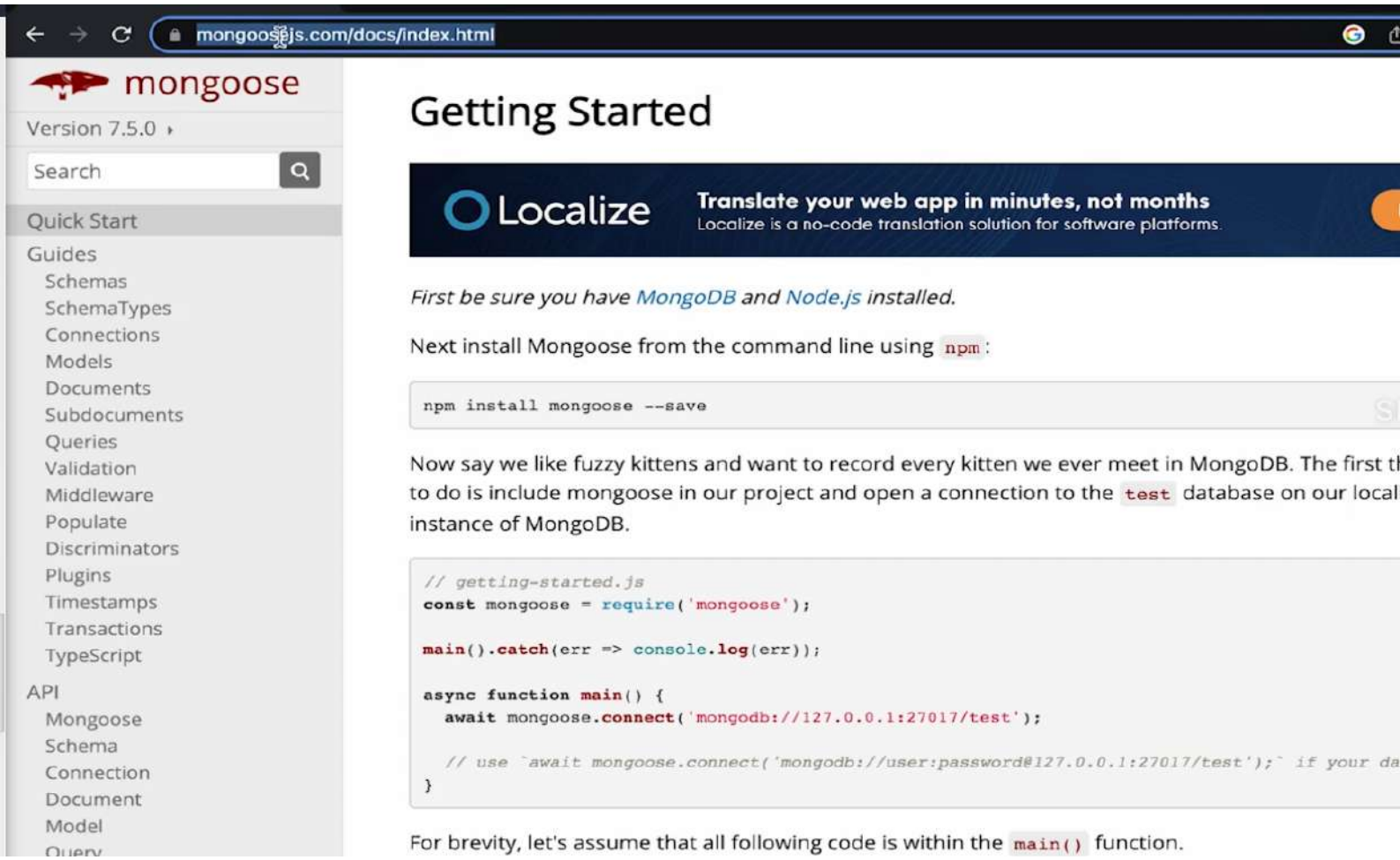
Installation

```
shradhakhapra@Shradhas-Air ~ % sudo brew services start mongodb-community@7.0
```

```
shradhakhapra@Shradhas-Air ~ % mongosh
```


test> show dbs
admin 40.00 KiB
config 72.00 KiB
local 72.00 KiB
test>

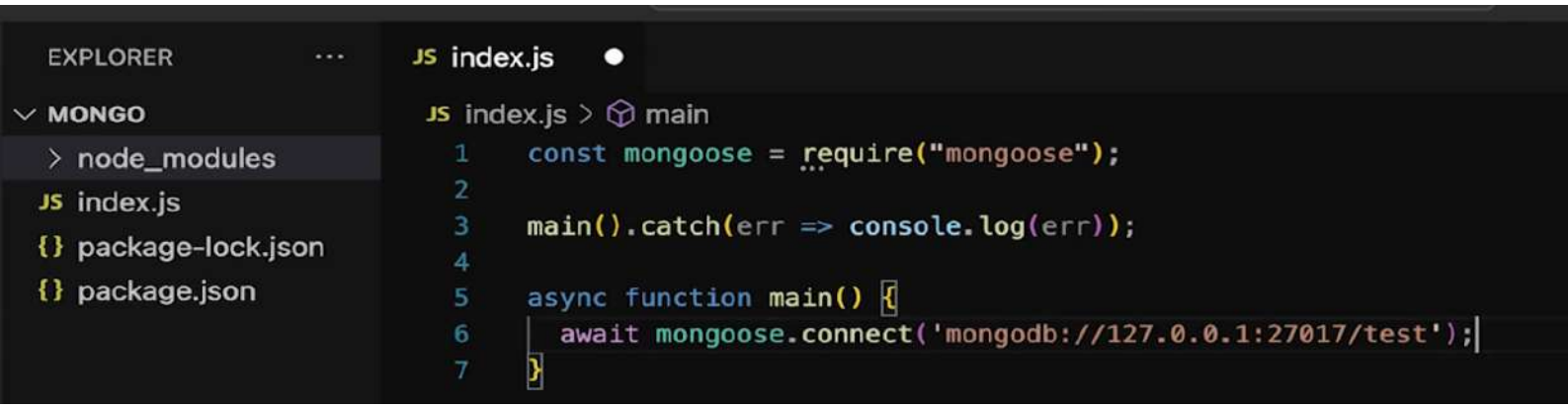

MONGO
package.json
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
"name": "mongo",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC"
}
OUTLINE
TIMELINE
shradhakhapra@Shradhas-MacBook-Air Mongo % npm i mongoose

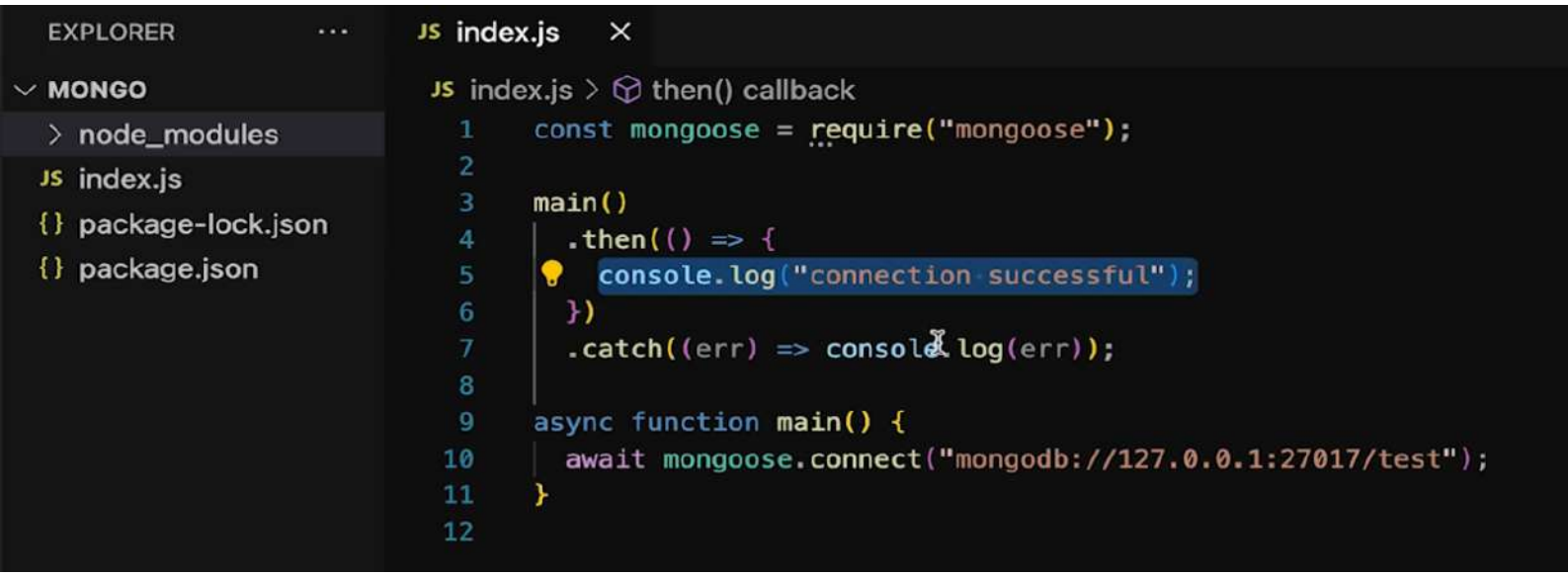

mongoosejs.com/docs/index.html
mongoose
Version 7.5.0
Search
Quick Start
Guides
Schemas
SchemaTypes
Connections
Models
Documents
Subdocuments
Queries
Validation
Middleware
Populate
Discriminators
Plugins
Timestamps
Transactions
TypeScript
API
Mongoose
Schema
Connection
Document
Model
Query
Getting Started
Localize
Translate your web app in minutes, not months
Localize is a no-code translation solution for software platforms.
First be sure you have MongoDB and Node.js installed.
Next install Mongoose from the command line using npm:
npm install mongoose --save
Now say we like fuzzy kittens and want to record every kitten we ever meet in MongoDB. The first th
to do is include mongoose in our project and open a connection to the test database on our local
instance of MongoDB.
// getting-started.js
const mongoose = require('mongoose');
main().catch(err => console.log(err));
async function main() {
await mongoose.connect('mongodb://127.0.0.1:27017/test');
// use `await mongoose.connect('mongodb://user:password@127.0.0.1:27017/test');` if your da
}
For brevity, let's assume that all following code is within the main() function.


index.js
index.js > url
1 const mongoose = require("mongoose");
2
3 let url = "https://localhost:8080/"
4
5 mongoose.connect("mongodb://127.0.0.1:27017/test");
6

```
mongoose.connect("mongodb://127.0.0.1:27017/instagram");
```


index.js
index.js > ...
1 const mongoose = require("mongoose");
2
3 mongoose.connect("mongodb://127.0.0.1:27017/test");
4


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > main
1 const mongoose = require("mongoose");
2
3 main().catch(err => console.log(err));
4
5 async function main() {
6 await mongoose.connect('mongodb://127.0.0.1:27017/test');
7 }


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > then() callback
const mongoose = require("mongoose");
main()
.then(() => {
console.log("connection successful");
})
.catch((err) => console.log(err));
async function main() {
await mongoose.connect("mongodb://127.0.0.1:27017/test");
}

```
^C
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
```

Schema

Schema defines the shape of the documents within that collection.

```
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number,
});
```

Defining your schema

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection.


```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const blogSchema = new Schema({
  title: String, // String is shorthand for {type: String}
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

The permitted SchemaTypes are:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array
- Decimal128
- Map
- UUID

Read more about SchemaTypes here

Models

Model in mongoose is a class with which we construct documents.

```
const User = mongoose.model("User", userSchema);
```

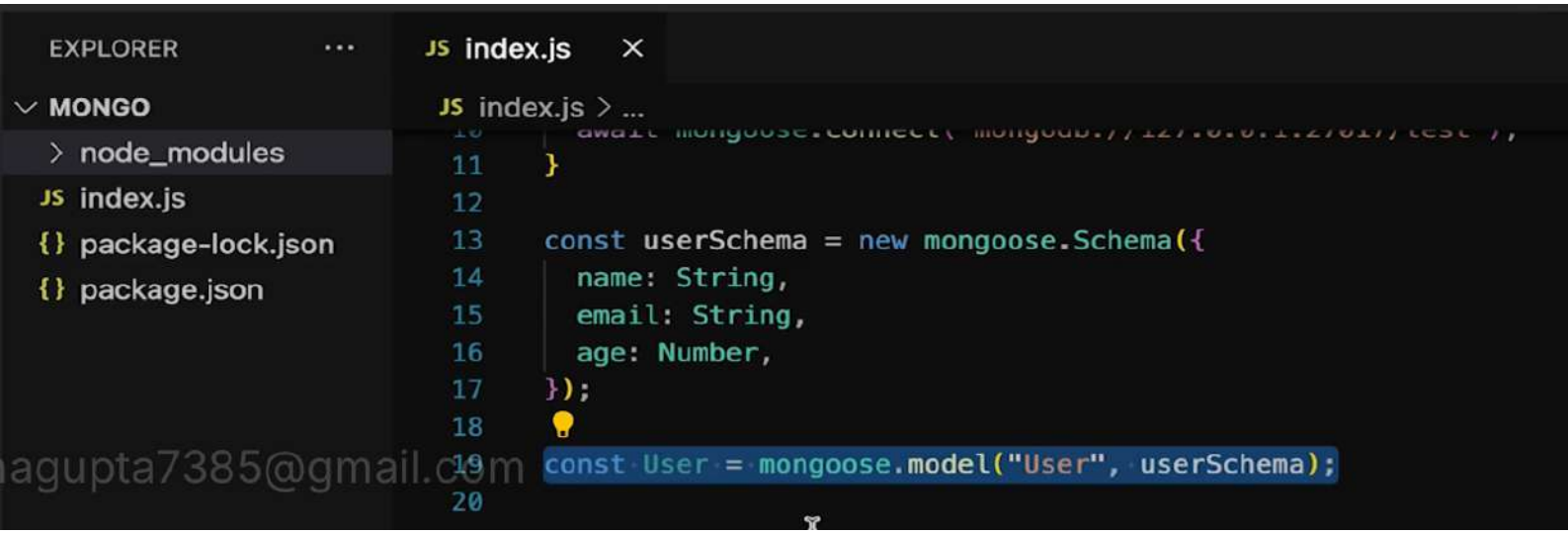
Models

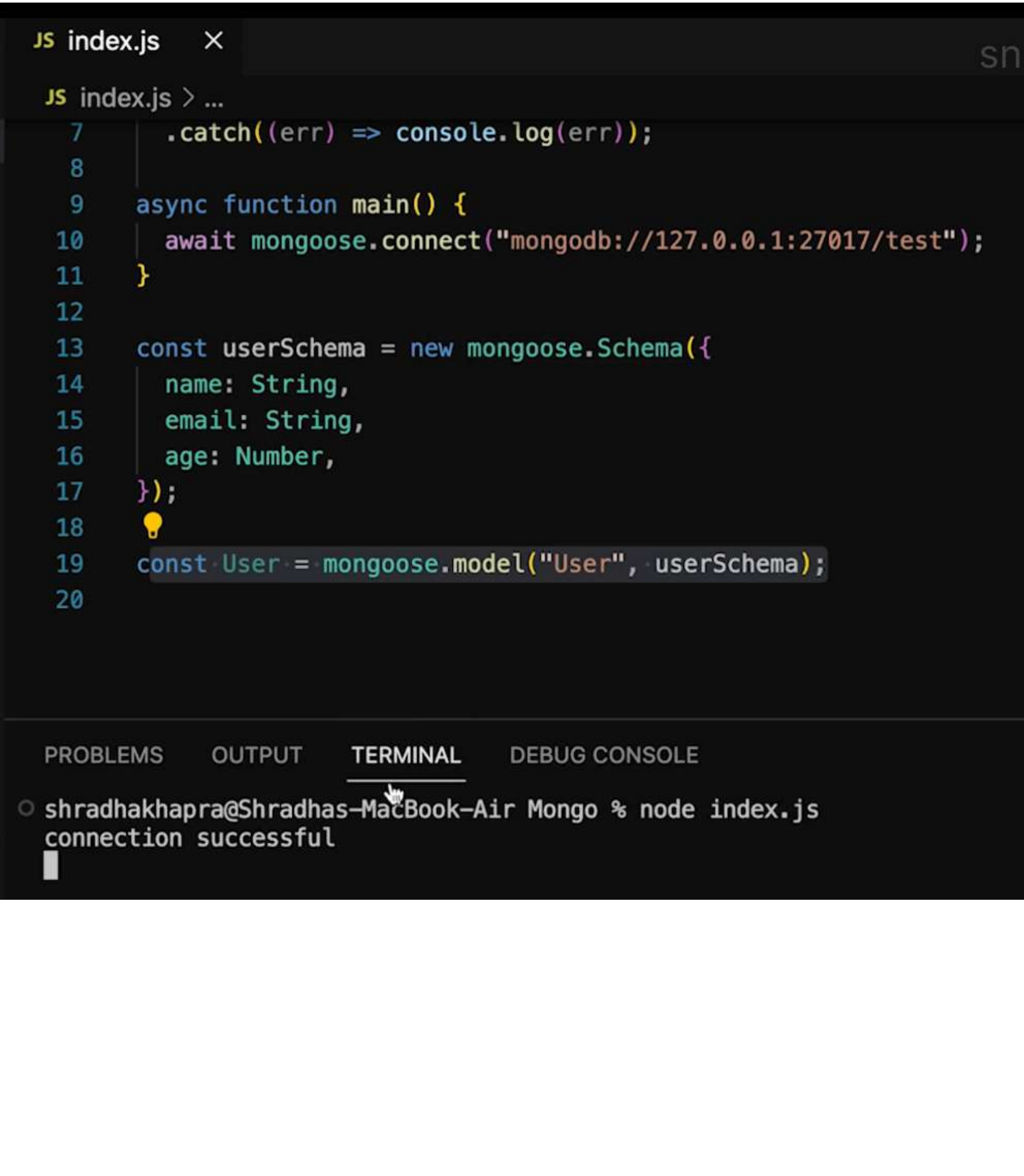
Model in mongoose is a class with which we construct documents.

```
const User = mongoose.model("User", userSchema);
```

Model

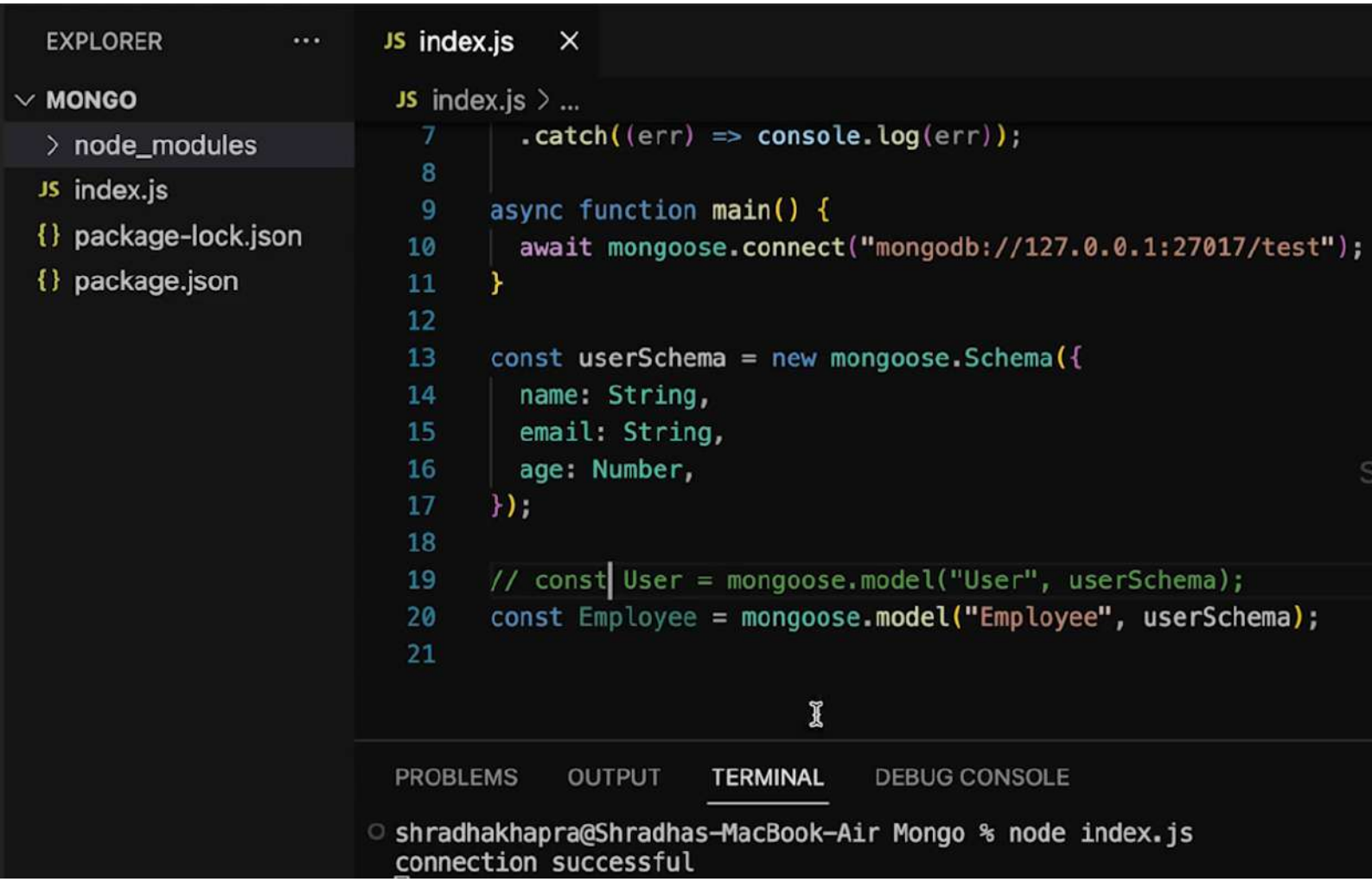
collection


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > ...
}
const userSchema = new mongoose.Schema({
name: String,
email: String,
age: Number,
});
const User = mongoose.model("User", userSchema);


index.js
index.js > ...
.catch((err) => console.log(err));
async function main() {
await mongoose.connect("mongodb://127.0.0.1:27017/test");
}
const userSchema = new mongoose.Schema({
name: String,
email: String,
age: Number,
});
const User = mongoose.model("User", userSchema);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful


test> show dbs
admin 40.00 KiB
config 72.00 KiB
local 72.00 KiB
test> show collections
users
test>


User → users
Product → products


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > ...
.catch((err) => console.log(err));
async function main() {
await mongoose.connect("mongodb://127.0.0.1:27017/test");
}
const userSchema = new mongoose.Schema({
name: String,
email: String,
age: Number,
});
// const User = mongoose.model("User", userSchema);
const Employee = mongoose.model("Employee", userSchema);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful

```
test> show dbs
admin    40.00 KiB
config   72.00 KiB
local    72.00 KiB
test> show collections
users
test> show collections
employees
users
test>
```

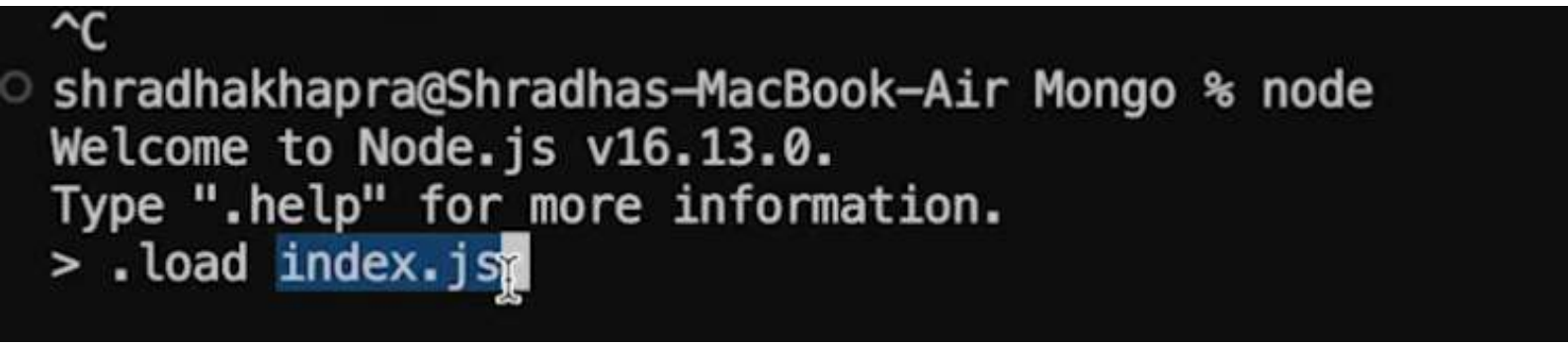
INSERT

Inserting One

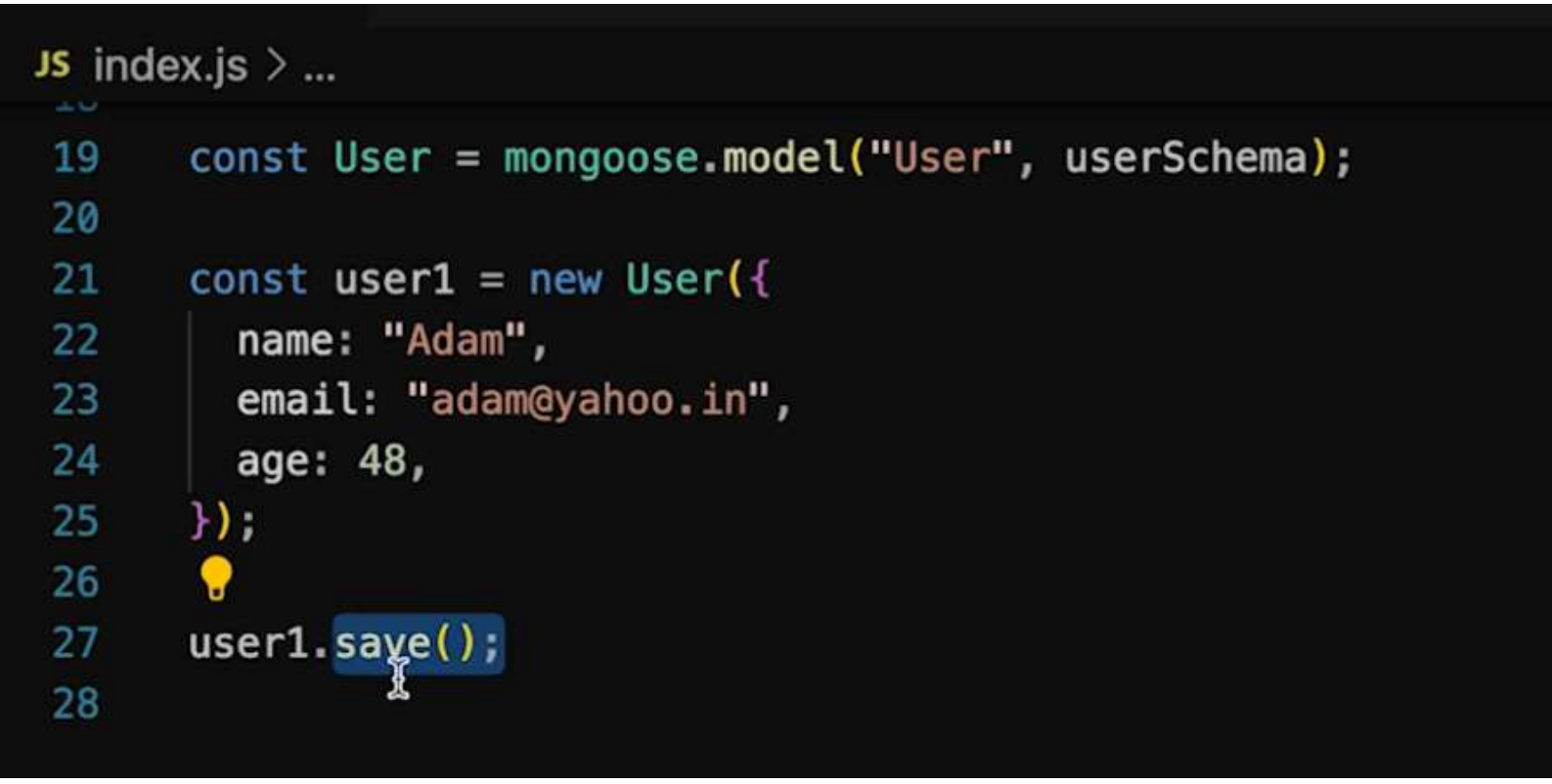
```
const user1 = new User({ name: "Adam", email: "adam@yahoo.in", age: 48 });
const user2 = new User({ name: "Eve", email: "eve@google.com", age: 48 });
```

```
user1.save( )  //to save in DB

user2.save( )  //to save in DB
```

```
admin    40.00 KiB
config   72.00 KiB
local    72.00 KiB
test> show collections
users
test> show collections
employees
users
test> show dbs
admin     40.00 KiB
config   108.00 KiB
local     72.00 KiB
test      16.00 KiB
test> db.users.find()

test>
```


^C
shradhakhapra@Shradhas-MacBook-Air Mongo % node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
> .load index.js

```
userer1
{
  name: 'Adam',
  email: 'adam@yahoo.in',
  age: 48,
  _id: new ObjectId("6500b931783aa6220a17a9b9")
}
> user1
{
  name: 'Adam',
  email: 'adam@yahoo.in',
  age: 48,
  _id: new ObjectId("6500b931783aa6220a17a9b9")
}
>
```


JS index.js > ...
19 const User = mongoose.model("User", userSchema);
20
21 const user1 = new User({
22 name: "Adam",
23 email: "adam@yahoo.in",
24 age: 48,
25 });
26
27 user1.save();
28

Ids

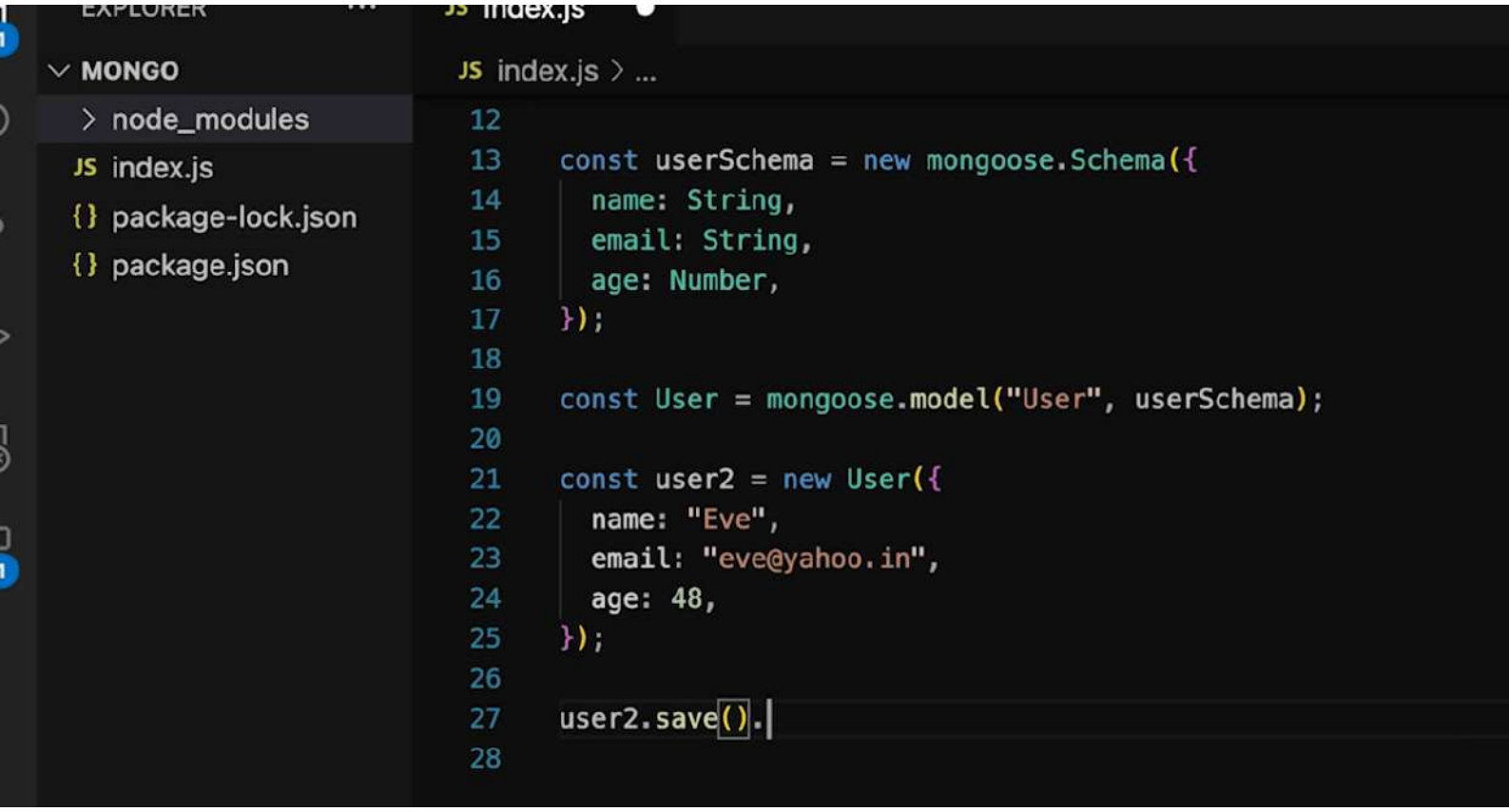
By default, Mongoose adds an `_id` property to your schemas.

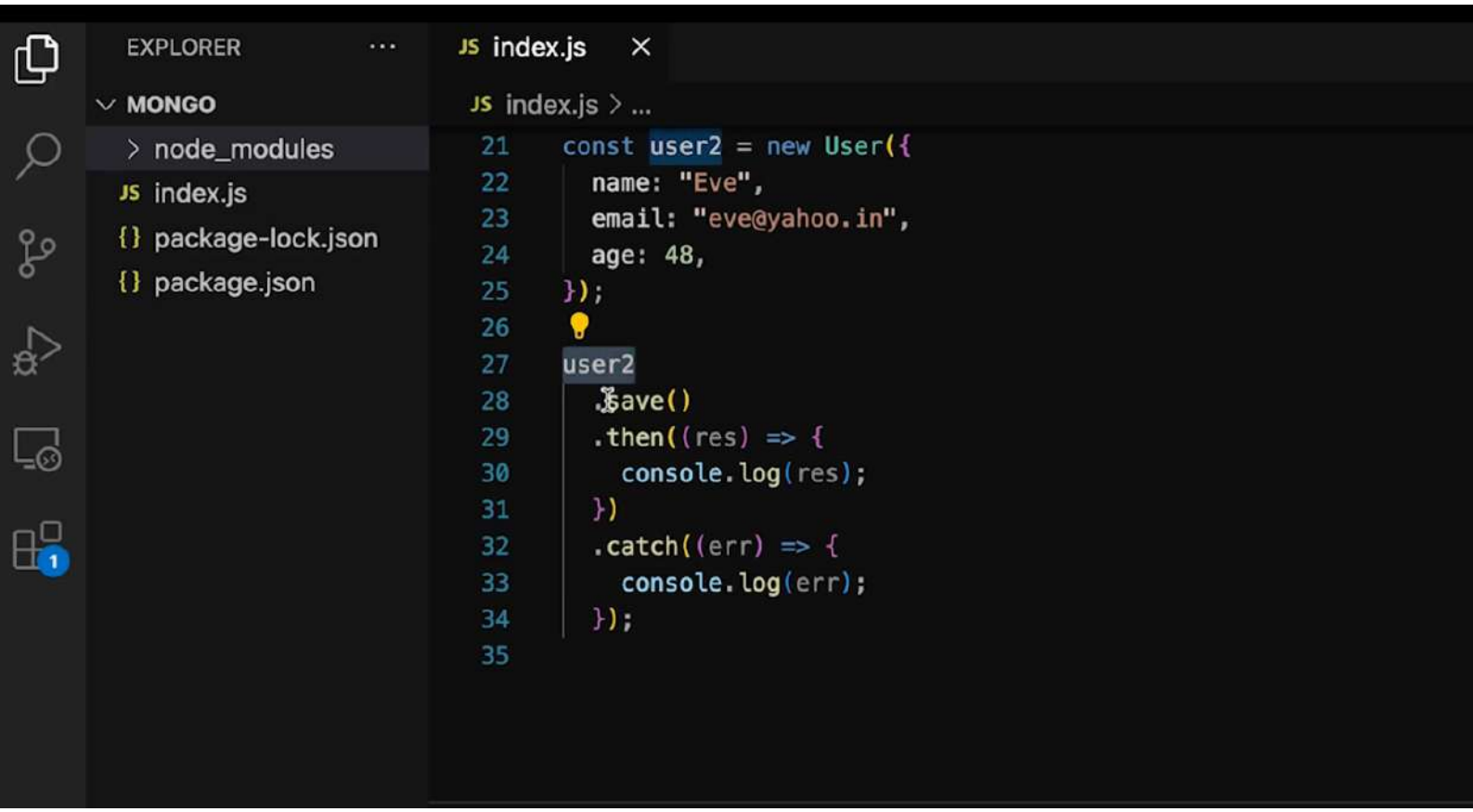
```
const schema = new Schema();

schema.path('_id'); // ObjectId { ... }
```

When you create a new document with the automatically added `_id` property, Mongoose creates a new `_id` of type ObjectId to your document.

```
const Model = mongoose.model('Test', schema);

const doc = new Model();
doc._id instanceof mongoose.Types.ObjectId; // true
```

```
test> db.users.find()

test> db.users.find()
[
  {
    _id: ObjectId("6500ba1c9e9c18aaaaa51542"),
    name: 'Adam',
    email: 'adam@yahoo.in',
    age: 48,
    __v: 0
  }
]
test>
```


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > ...
12
13 const userSchema = new mongoose.Schema({
14 name: String,
15 email: String,
16 age: Number,
17 });
18
19 const User = mongoose.model("User", userSchema);
20
21 const user2 = new User({
22 name: "Eve",
23 email: "eve@yahoo.in",
24 age: 48,
25 });
26
27 user2.save().
28


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > ...
const user2 = new User({
name: "Eve",
email: "eve@yahoo.in",
age: 48,
});
user2
.save()
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});

```
27    user2
28    .save()
29    .then((res) => {
30      console.log(res);
31    })
32    .catch((err) => {
33      console.log(err);
34    });
35
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

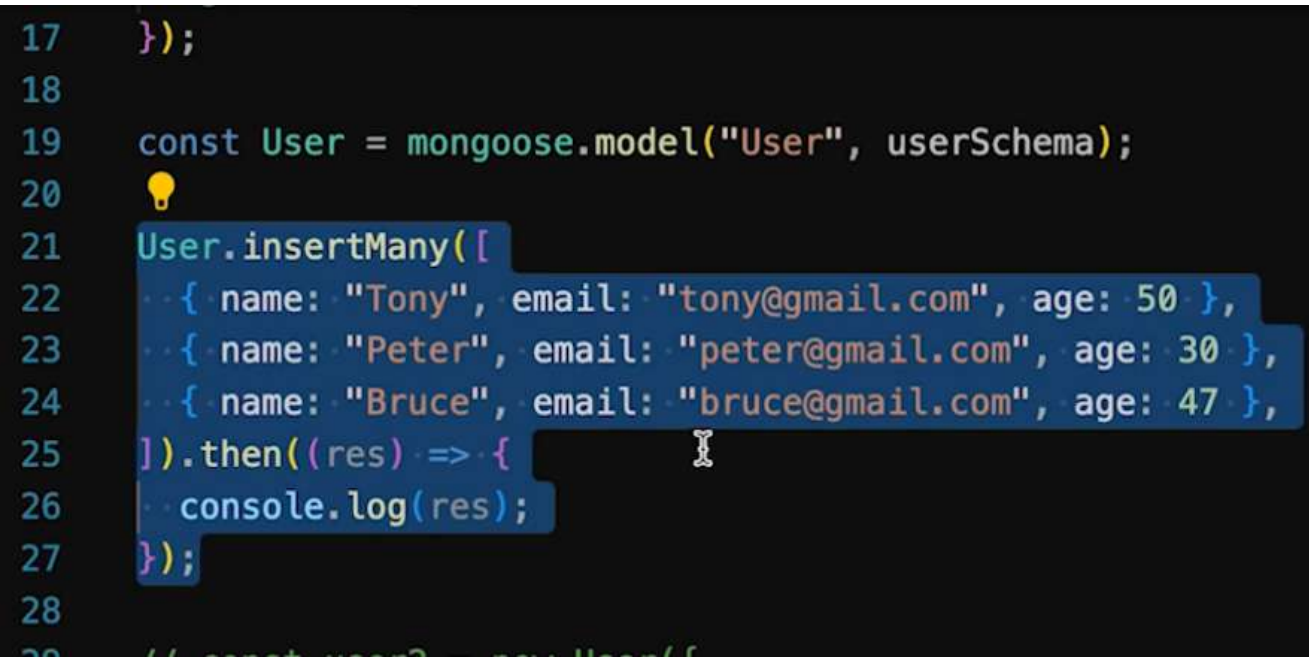
```
userer1
{
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
  name: 'Eve',
  email: 'eve@yahoo.in',
  age: 48,
  _id: new ObjectId("6500ba8a8ecc3e6eb6166393"),
  __v: 0
}
```

```
    __v: 0
  }
]
test> db.users.find()
[
  {
    _id: ObjectId("6500ba1c9e9c18aaaaa51542"),
    name: 'Adam',
    email: 'adam@yahoo.in',
    age: 48,
    __v: 0
  },
  {
    _id: ObjectId("6500ba8a8ecc3e6eb6166393"),
    name: 'Eve',
    email: 'eve@yahoo.in',
    age: 48,
    __v: 0
  }
]
test>
```

INSERT

Inserting Multiple

```
User.insertMany([
  { name: "Tony", email: "tony@gmail.com", age: 50 },
  { name: "Bruce", email: "bruce@gmail.com", age: 47 },
  { name: "Peter", email: "peter@gmail.com", age: 30 },
]).then((data) => {
  console.log(data);
});
```


});
const User = mongoose.model("User", userSchema);
User.insertMany([
{ name: "Tony", email: "tony@gmail.com", age: 50 },
{ name: "Peter", email: "peter@gmail.com", age: 30 },
{ name: "Bruce", email: "bruce@gmail.com", age: 47 },
]).then((res) => {
console.log(res);
});

```
    age: 48,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 50,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter',
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f5"),
    name: 'Bruce',
    email: 'bruce@gmail.com',
    age: 47,
    __v: 0
  }
]
```

Note

Mongoose uses Operation Buffering

Mongoose lets you start using your models immediately, without waiting for mongoose to establish a connection to MongoDB.

FIND

Model.find( ) //returns a Query Object (thennable)

***Mongoose Queries are not promises. But they have a .then( )**

```
User.find().then((data) => {
  console.log(data);
});
```

```
User.find({ age: { $gte: 47 } }).then((data) => {
  console.log(data);
});
```

- `Model.find()`
- `Model.findById()`
- `Model.findByIdAndDelete()`
- `Model.findByIdAndRemove()`
- `Model.findByIdAndUpdate()`
- `Model.findOne()`
- `Model.findOneAndDelete()`
- `Model.findOneAndRemove()`
- `Model.findOneAndReplace()`
- `Model.findOneAndUpdate()`
- `Model.hydrate()`
- `Model.init()`
- `Model.insertMany()`
- `Model.inspect()`

- `[options.translateAliases=null]` «Boolean» If set to `true`, translates any schema-defined aliases in `filter`, `projection`, `update`, and `distinct`. Throws an error if there are any conflicts where both alias and raw property are defined on the same object.

Returns:

- «Query»

See:

- field selection
- query casting

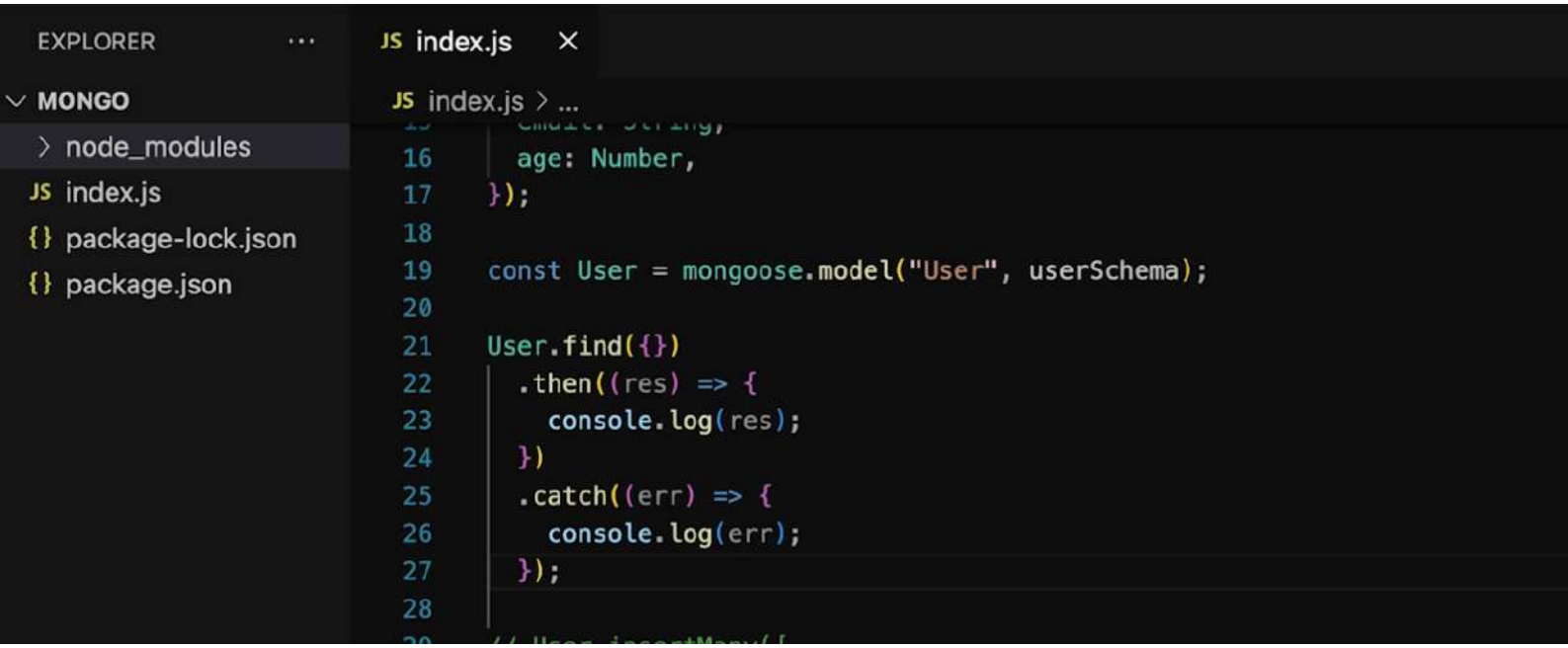
Finds documents.

Mongoose casts the `filter` to match the model's schema before the command is sent. See our query casting tutorial for more information on how Mongoose casts `filter`.

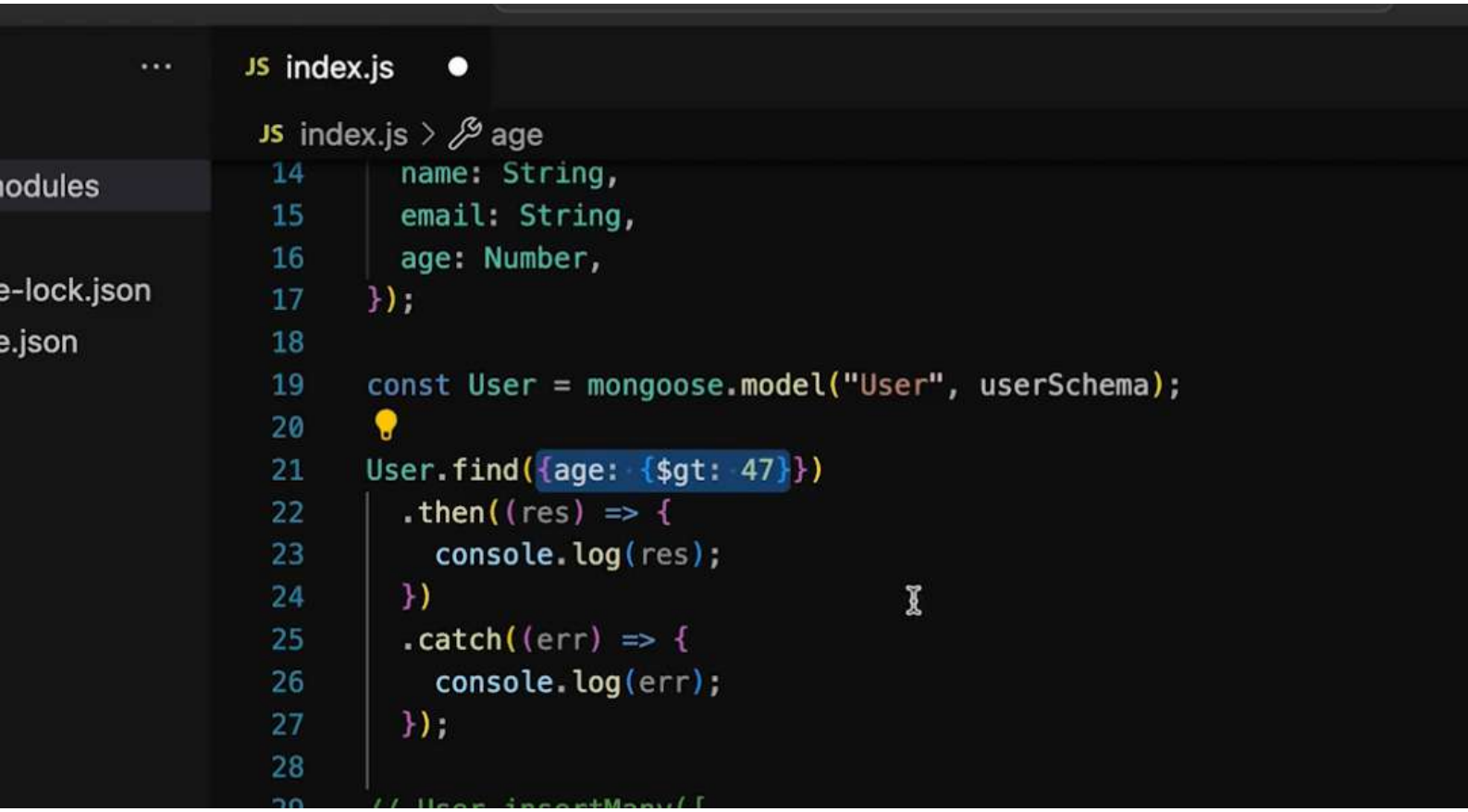
Example:

```
// find all documents
await MyModel.find({});

// find all documents named john and at least 18
await MyModel.find({ name: 'john', age: { $gte: 18 } }).exec();

// executes, name LIKE john and only selecting the "name" and "friends" fields
await MyModel.find({ name: /john/i }, 'name friends').exec();

// passing options
await MyModel.find({ name: /john/i }, null, { skip: 10 }).exec();
```


JS index.js
JS index.js > ...
const User = mongoose.model("User", userSchema);
User.find({}).then((res) => {
console.log(res);
})


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > ...
age: Number,
});
const User = mongoose.model("User", userSchema);
User.find({})
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});

```
    _id: new ObjectId("6500ba8a8ecc3e6eb6166393"),
    name: 'Eve',
    email: 'eve@yahoo.in',
    age: 48,
    __v: 0
  },
  {
    _id: new ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 50,
    __v: 0
  },
  {
    _id: new ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter',
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  },
  {
    _id: new ObjectId("6500bb8411e600b2e42db9f5"),
    name: 'Bruce',
    email: 'bruce@gmail.com',
    age: 47,
    __v: 0
  }
]
```

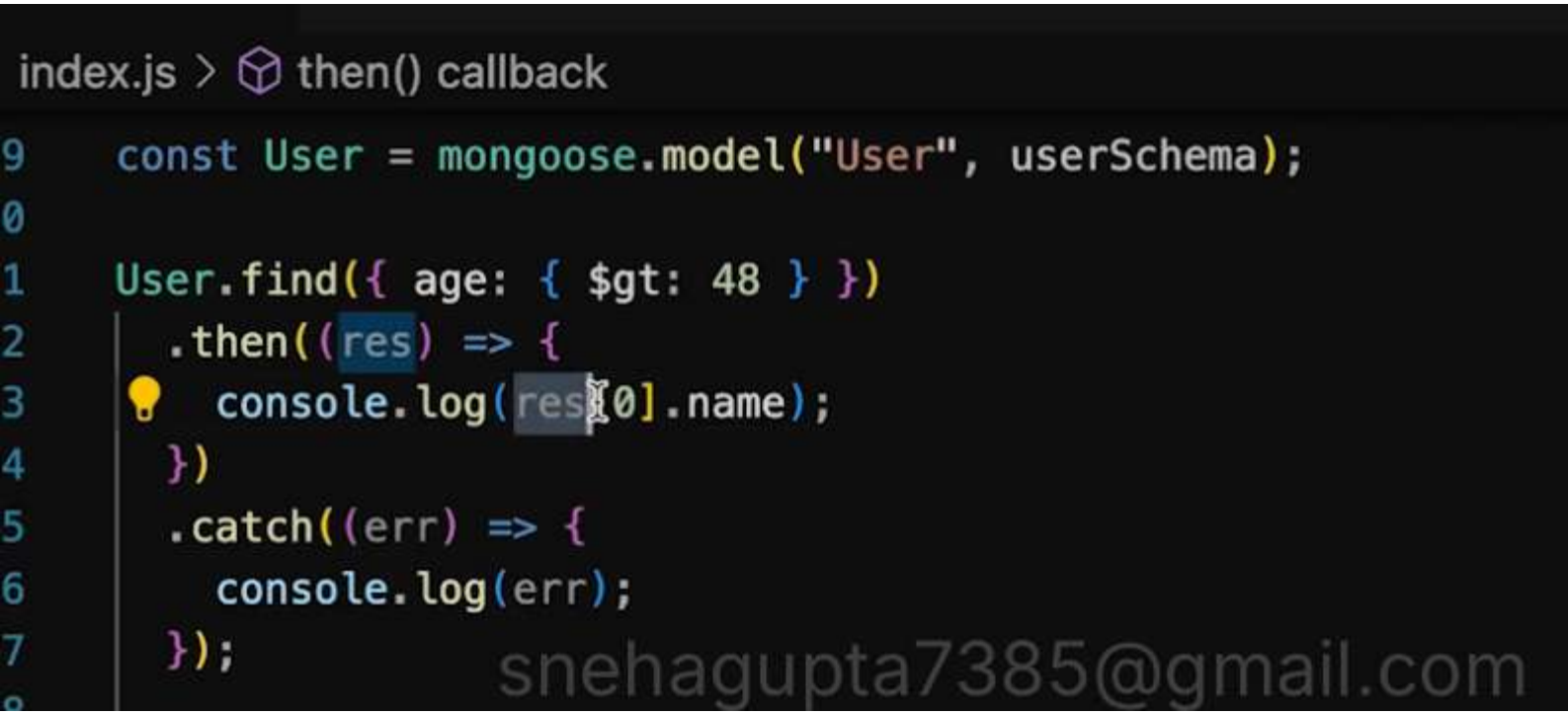

index.js
index.js > age
odules
e-lock.json
e.json
name: String,
email: String,
age: Number,
});
const User = mongoose.model("User", userSchema);
User.find({age: {$gt: 47}})
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});

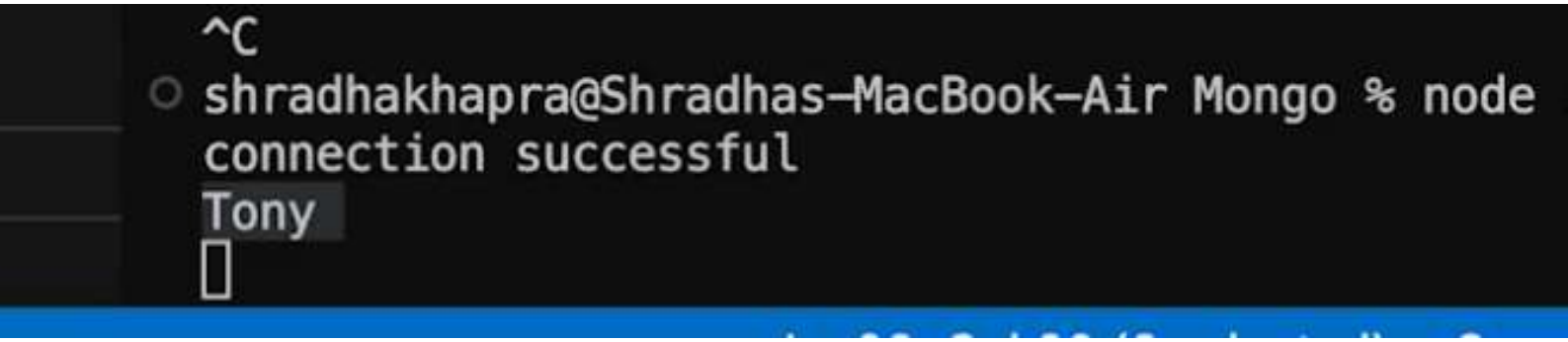
```
    age: 30,
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
[
  {
    _id: new ObjectId("6500ba1c9e9c18aaaaa51542"),
    name: 'Adam',
    email: 'adam@yahoo.in',
    age: 48,
    __v: 0
  },
  {
    _id: new ObjectId("6500ba8a8ecc3e6eb6166393"),
    name: 'Eve',
    email: 'eve@yahoo.in',
    age: 48,
    __v: 0
  },
  {
    _id: new ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 50,
    __v: 0
  }
]
```


19 const User = mongoose.model("User", userSchema);
20
21 User.find({ age: { $gt: 48 } })
22 .then((res) => {
23 console.log(res[0]);
24 })
25 .catch((err) => {
26 console.log(err);
27 });

```
C
hradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
onnection successful

 _id: new ObjectId("6500bb8411e600b2e42db9f3"),
 name: 'Tony',
 email: 'tony@gmail.com',
 age: 50,
 __v: 0
```

Ln 23, Col 28 Spaces: 4 UTF-8 LF


index.js > then() callback
const User = mongoose.model("User", userSchema);
User.find({ age: { $gt: 48 } })
.then((res) => {
console.log(res[0].name);
})
.catch((err) => {
console.log(err);
});
snehagupta7385@gmail.com


^C
shradhakhapra@Shradhas-MacBook-Air Mongo % node
connection successful
Tony

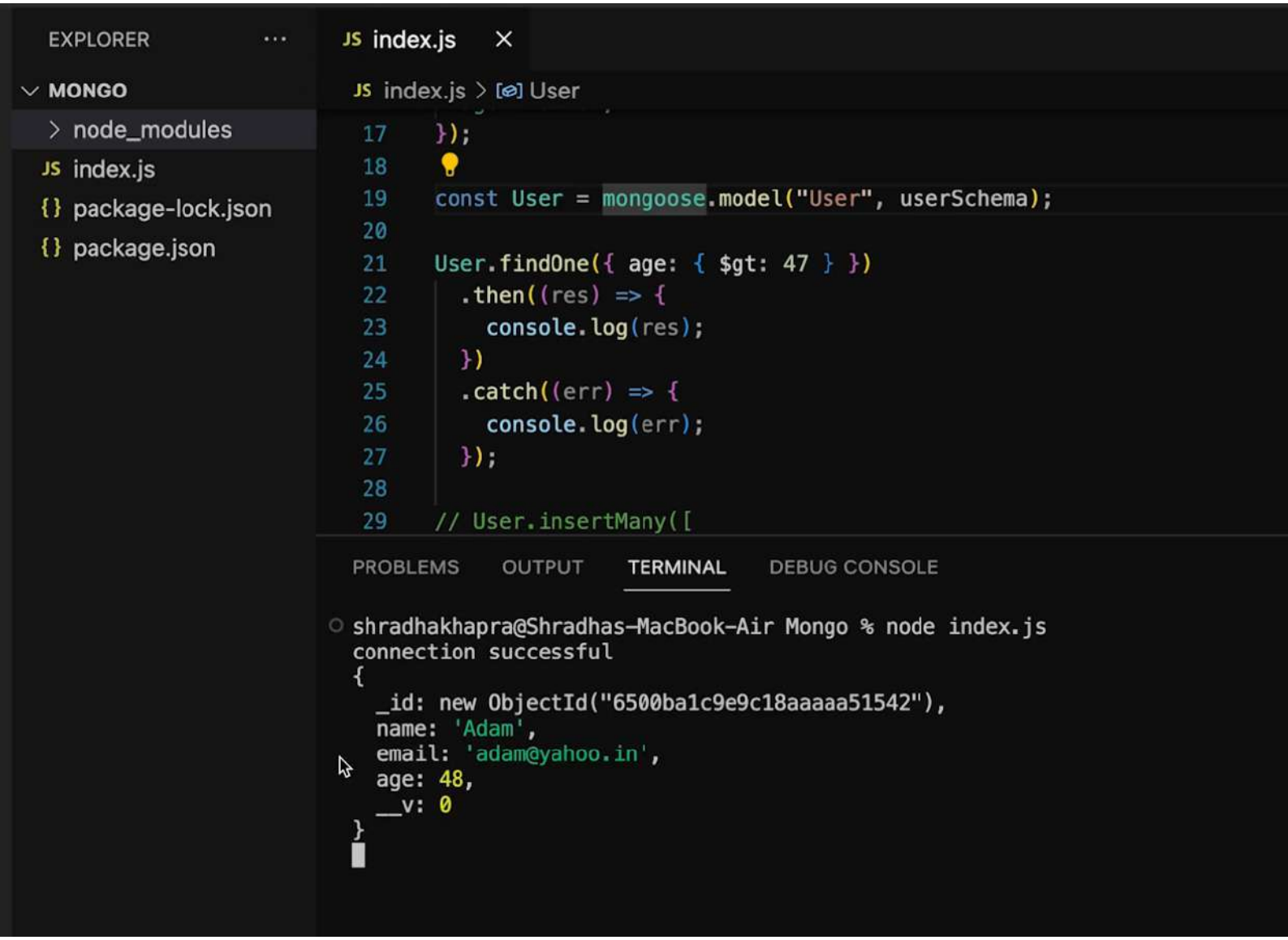
FIND

Model.findOne() *//returns a single result*

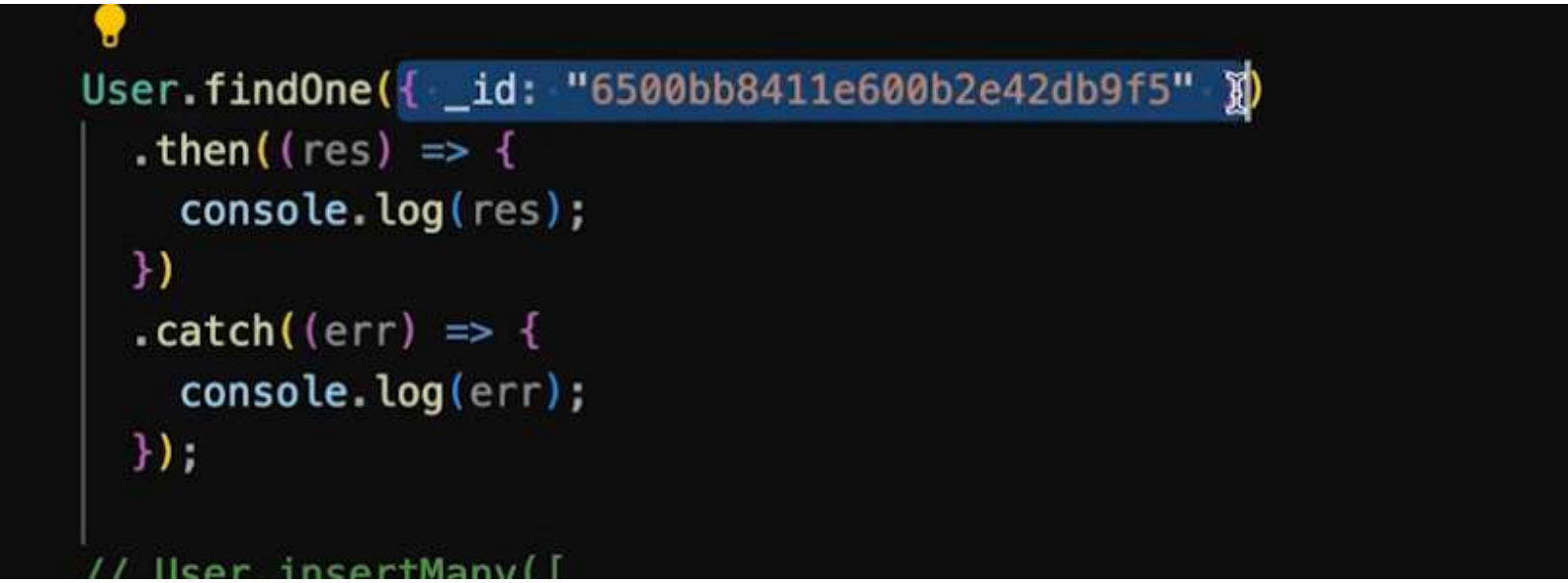
```
User.findOne({ age: { $gte: 47 } }).then((data) => {
  console.log(data);
});
```

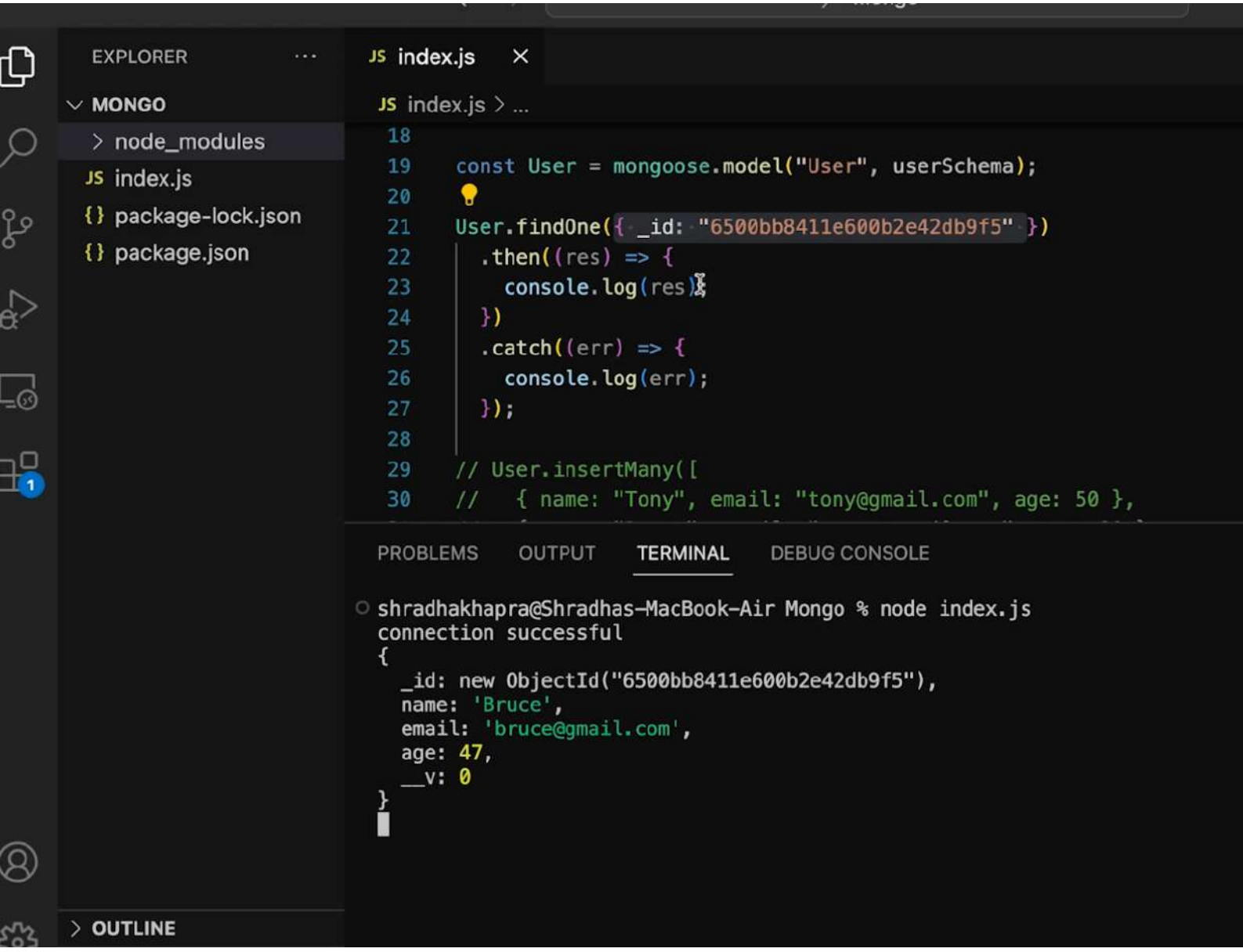
snehagupta7385@gmail.com

Model.findById() *//commonly used*


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > User
});
const User = mongoose.model("User", userSchema);
User.findOne({ age: { $gt: 47 } })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});
// User.insertMany([
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500ba1c9e9c18aaaaa51542"),
name: 'Adam',
email: 'adam@yahoo.in',
age: 48,
__v: 0
}

```
  age: 48,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f3"),
  name: 'Tony',
  email: 'tony@gmail.com',
  age: 50,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f4"),
  name: 'Peter',
  email: 'peter@gmail.com',
  age: 30,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f5"),
  name: 'Bruce',
  email: 'bruce@gmail.com',
  age: 47,
  __v: 0
}
```


User.findOne({ _id: "6500bb8411e600b2e42db9f5" })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
OUTLINE
index.js
index.js > ...
18
19 const User = mongoose.model("User", userSchema);
20
21 User.findOne({ _id: "6500bb8411e600b2e42db9f5" })
22 .then((res) => {
23 console.log(res);
24 })
25 .catch((err) => {
26 console.log(err);
27 });
28
29 // User.insertMany([
30 // { name: "Tony", email: "tony@gmail.com", age: 50 },
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
email: 'bruce@gmail.com',
age: 47,
__v: 0
}

`Model.findById()`

Parameters:

- `id` «Any» value of `_id` to query by
- `[projection]` «Object|String|Array[String]» optional fields to return, see `Query.prototype.select()`
- `[options]` «Object» optional see `Query.prototype.setOptions()`

Returns:

- «Query»

See:

- field selection
- lean queries
- findById in Mongoose

Finds a single document by its _id field. `findById(id)` is almost* equivalent to `findOne({ _id: id })`. If you want to query by a document's `_id`, use `findById()` instead of `findOne()`.

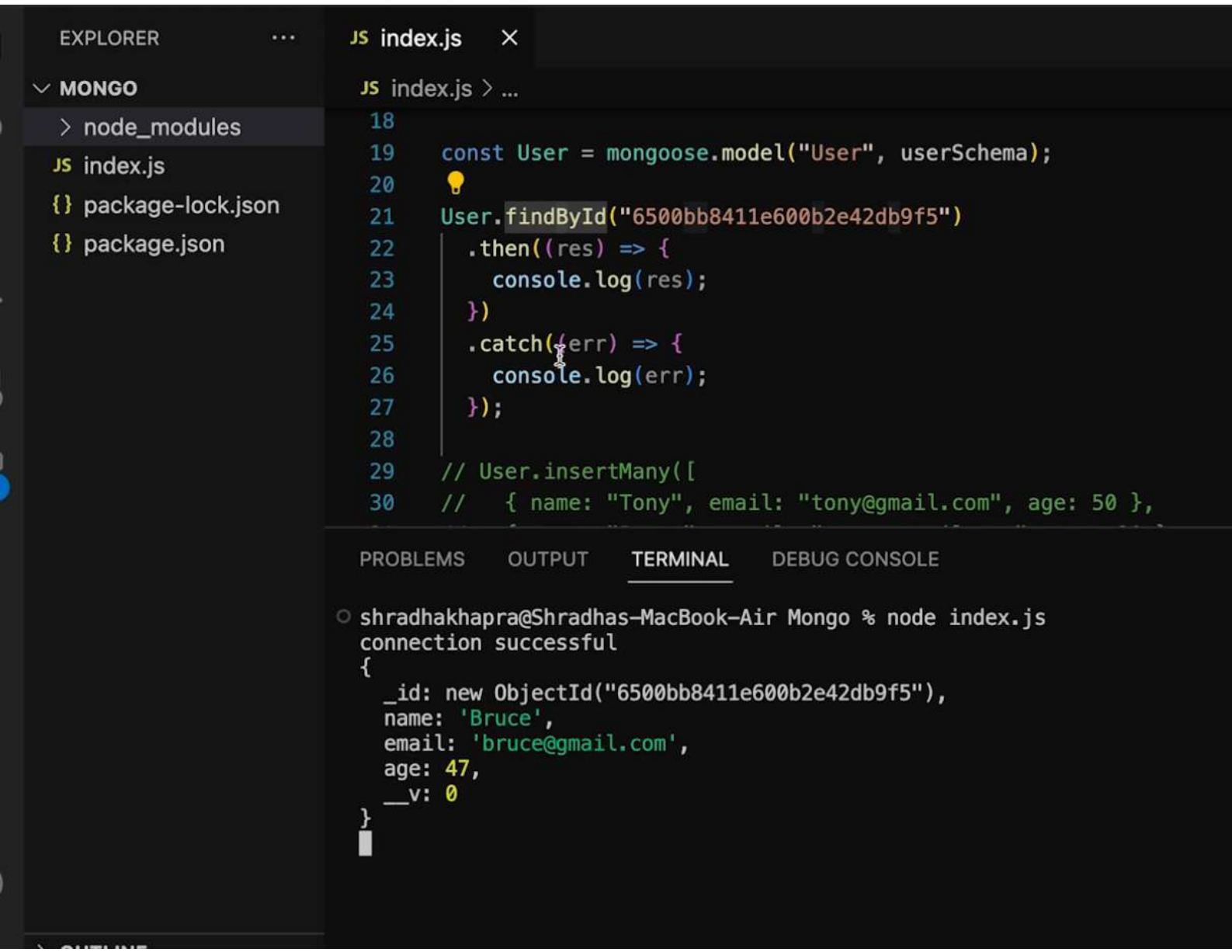
The `id` is cast based on the Schema before sending the command.

This function triggers the following middleware.

- `findOne()`

* Except for how it treats `undefined`. If you use `findOne()`, you'll see that `findOne(undefined)` and `findOne({ _id: undefined })` are equivalent to `findOne({})` and return arbitrary documents. However, mongoose translates `findById(undefined)` into `findOne({ _id: null })`.

Example:


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
index.js
index.js > ...
18
19 const User = mongoose.model("User", userSchema);
20
21 User.findById("6500bb8411e600b2e42db9f5")
22 .then((res) => {
23 console.log(res);
24 })
25 .catch((err) => {
26 console.log(err);
27 });
28
29 // User.insertMany([
30 // { name: "Tony", email: "tony@gmail.com", age: 50 },
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
email: 'bruce@gmail.com',
age: 47,
__v: 0
}

UPDATE

Model.updateOne()

```
User.updateOne({ name: "Bruce" }, { age: 49 }).then((res) => {
  console.log(res);
});
```

Model.updateMany()

```
User.updateMany({ age: { $gt: 45 } }, { age: 45 }).then((res) => {
  console.log(res);
});
```

- `Model.startSession()`
- `Model.syncIndexes()`
- `Model.translateAliases()`
- `Model.updateMany()`
- `Model.updateOne()`
- `Model.validate()`

`Model.updateOne()`

Parameters:

- `filter` «Object»
- `update` «Object|Array»
- `[options]` «Object» optional see `Query.prototype.setOptions()`
  - `[options.strict]` «Boolean|String» overwrites the schema's strict mode option
  - `[options.upsert=false]` «Boolean» if true, and no documents found, insert a new document
  - `[options.writeConcern=null]` «Object» sets the write concern for replica sets. Overrides the schema-level write concern
  - `[options.timestamps=null]` «Boolean» If set to `false` and schema-level timestamps are enabled, skip timestamps for this update. Note that this allows you to overwrite timestamps. Does nothing if schema-level timestamps are not set.
  - `[options.translateAliases=null]` «Boolean» If set to `true`, translates any schema-defined aliases in `filter`, `projection`, `update`, and `distinct`. Throws an error if there are any conflicts where both alias and raw property are defined on the same object.

Returns:

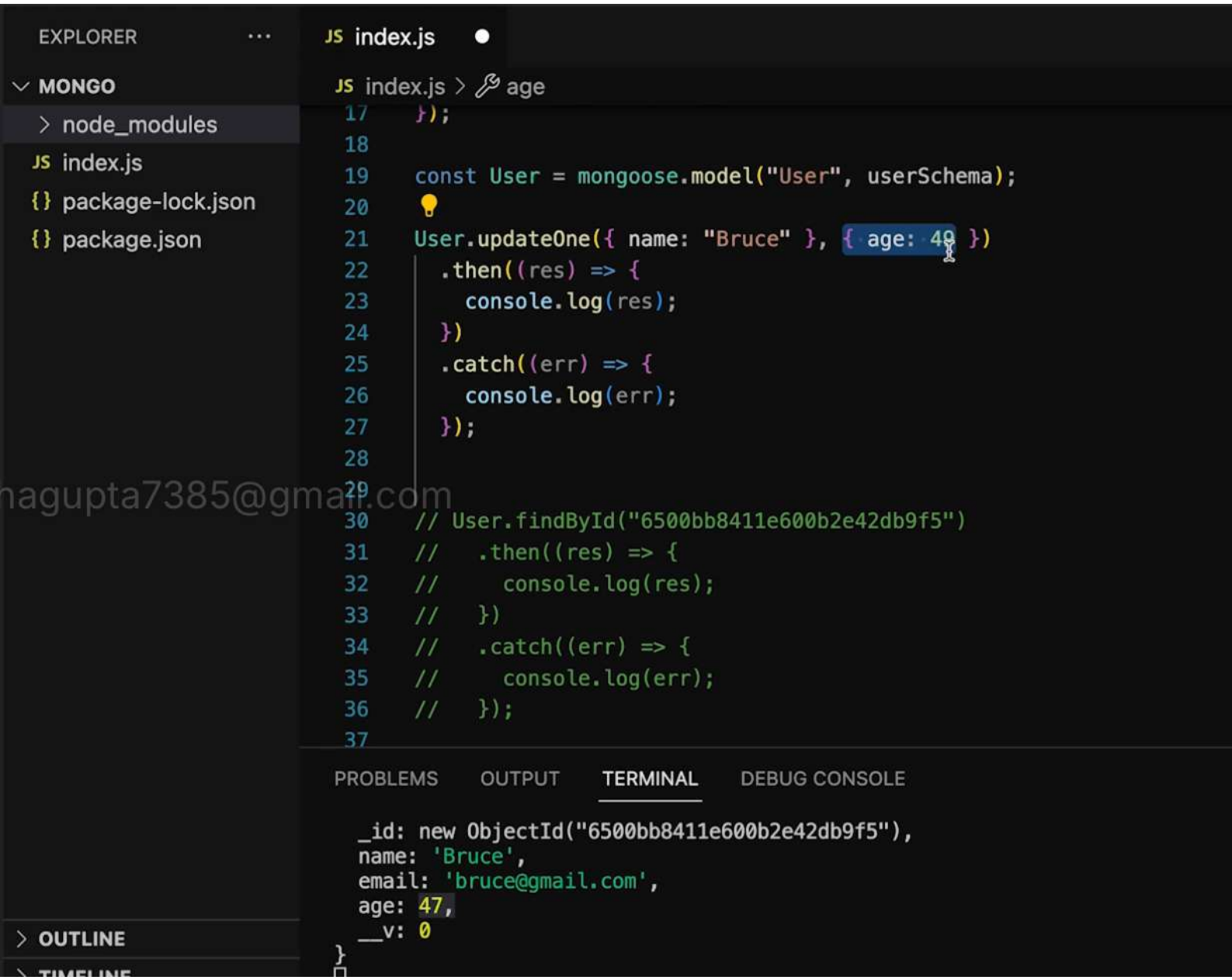
- «Query»

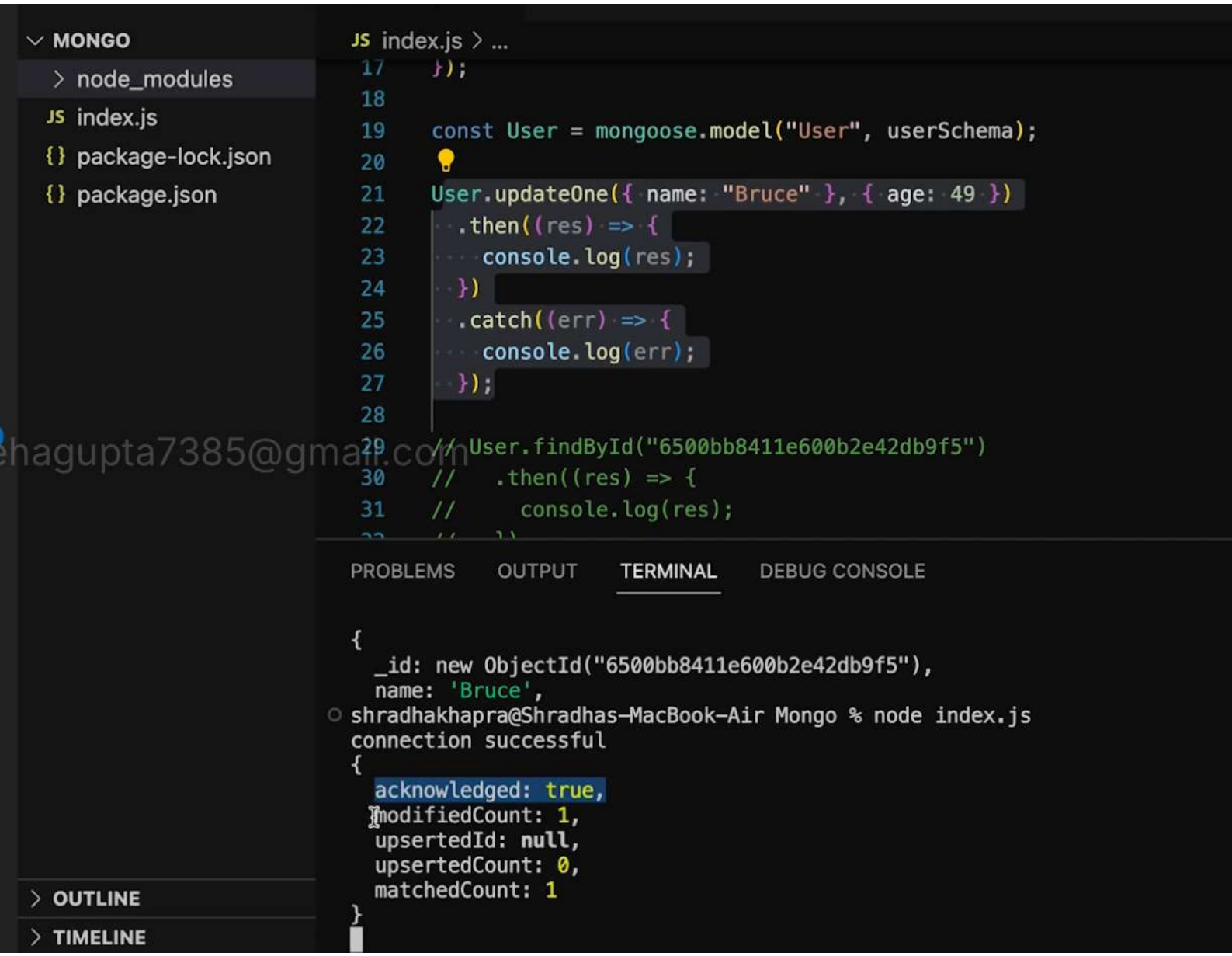
See:

- Query docs
- MongoDB docs
- UpdateResult

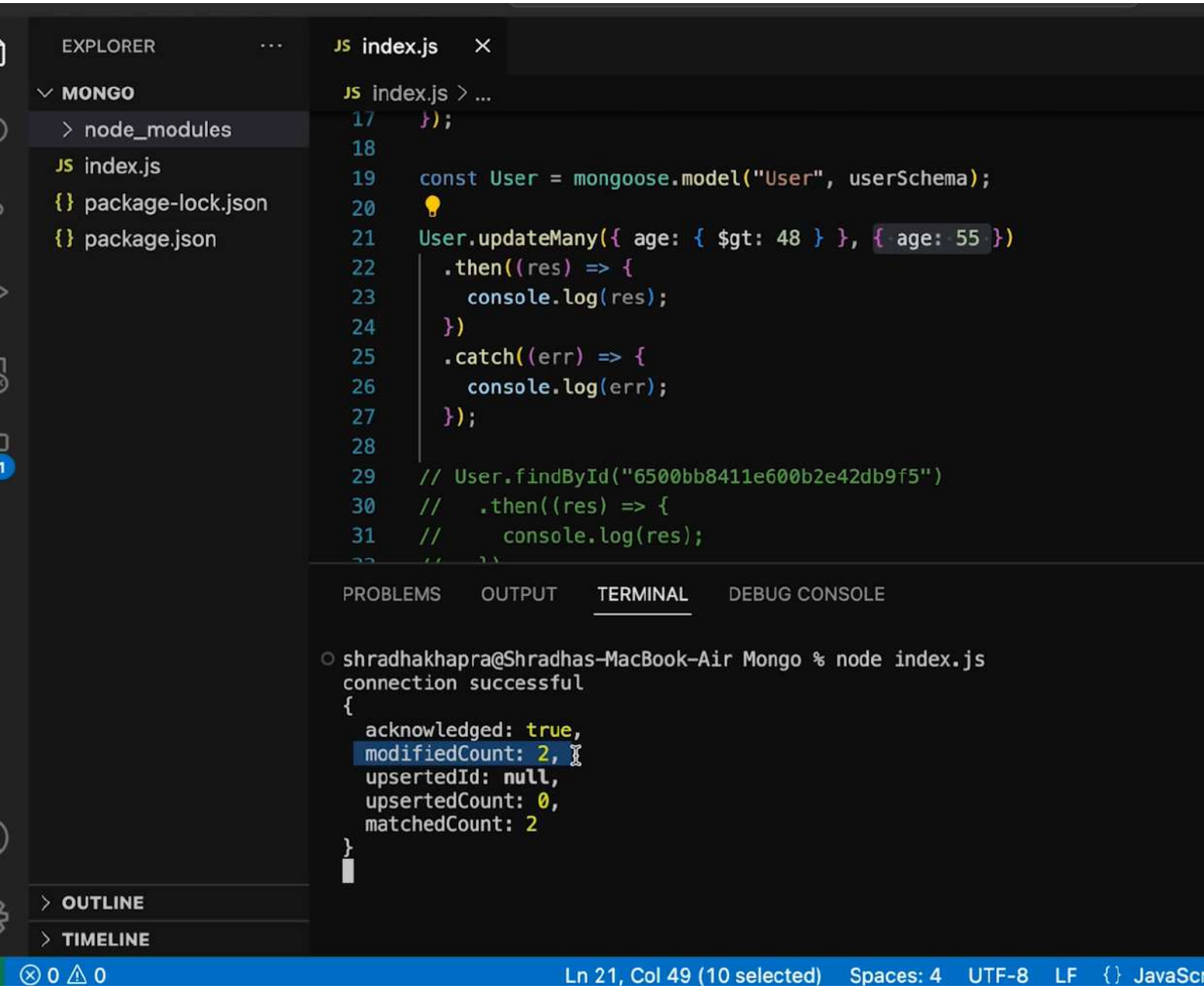
Update *only* the first document that matches `filter`.

- Use `replaceOne()` if you want to overwrite an entire document rather than using atomic operators like


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > age
});
const User = mongoose.model("User", userSchema);
User.updateOne({ name: "Bruce" }, { age: 40 })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});
// User.findById("6500bb8411e600b2e42db9f5")
// .then((res) => {
// console.log(res);
// })
// .catch((err) => {
// console.log(err);
// });
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
email: 'bruce@gmail.com',
age: 47,
__v: 0
}
OUTLINE


MONGO
node_modules
index.js
package-lock.json
package.json
OUTLINE
TIMELINE
index.js > ...
});
const User = mongoose.model("User", userSchema);
User.updateOne({ name: "Bruce" }, { age: 49 })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});
// User.findById("6500bb8411e600b2e42db9f5")
// .then((res) => {
// console.log(res);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
{
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
acknowledged: true,
modifiedCount: 1,
upsertedId: null,
upsertedCount: 0,
matchedCount: 1
}

```
  age: 48,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f3"),
  name: 'Tony',
  email: 'tony@gmail.com',
  age: 50,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f4"),
  name: 'Peter',
  email: 'peter@gmail.com',
  age: 30,
  __v: 0
},
{
  _id: ObjectId("6500bb8411e600b2e42db9f5"),
  name: 'Bruce',
  email: 'bruce@gmail.com',
  age: 49,
  __v: 0
}
```


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
OUTLINE
TIMELINE
index.js
index.js > ...
17 });
18
19 const User = mongoose.model("User", userSchema);
20
21 User.updateMany({ age: { $gt: 48 } }, { age: 55 })
22 .then((res) => {
23 console.log(res);
24 })
25 .catch((err) => {
26 console.log(err);
27 });
28
29 // User.findById("6500bb8411e600b2e42db9f5")
30 // .then((res) => {
31 // console.log(res);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
acknowledged: true,
modifiedCount: 2,
upsertedId: null,
upsertedCount: 0,
matchedCount: 2
}
0 0
Ln 21, Col 49 (10 selected)
Spaces: 4
UTF-8
LF

```
    age: 48,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 55,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter',
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f5"),
    name: 'Bruce',
    email: 'bruce@gmail.com',
    age: 55,
    __v: 0
  }
]
```

UPDATE

Model.findOneAndUpdate()

```
User.updateMany({ age: { $gt: 45 } }, { age: 45 }).then((res) => {
  console.log(res);
});
```

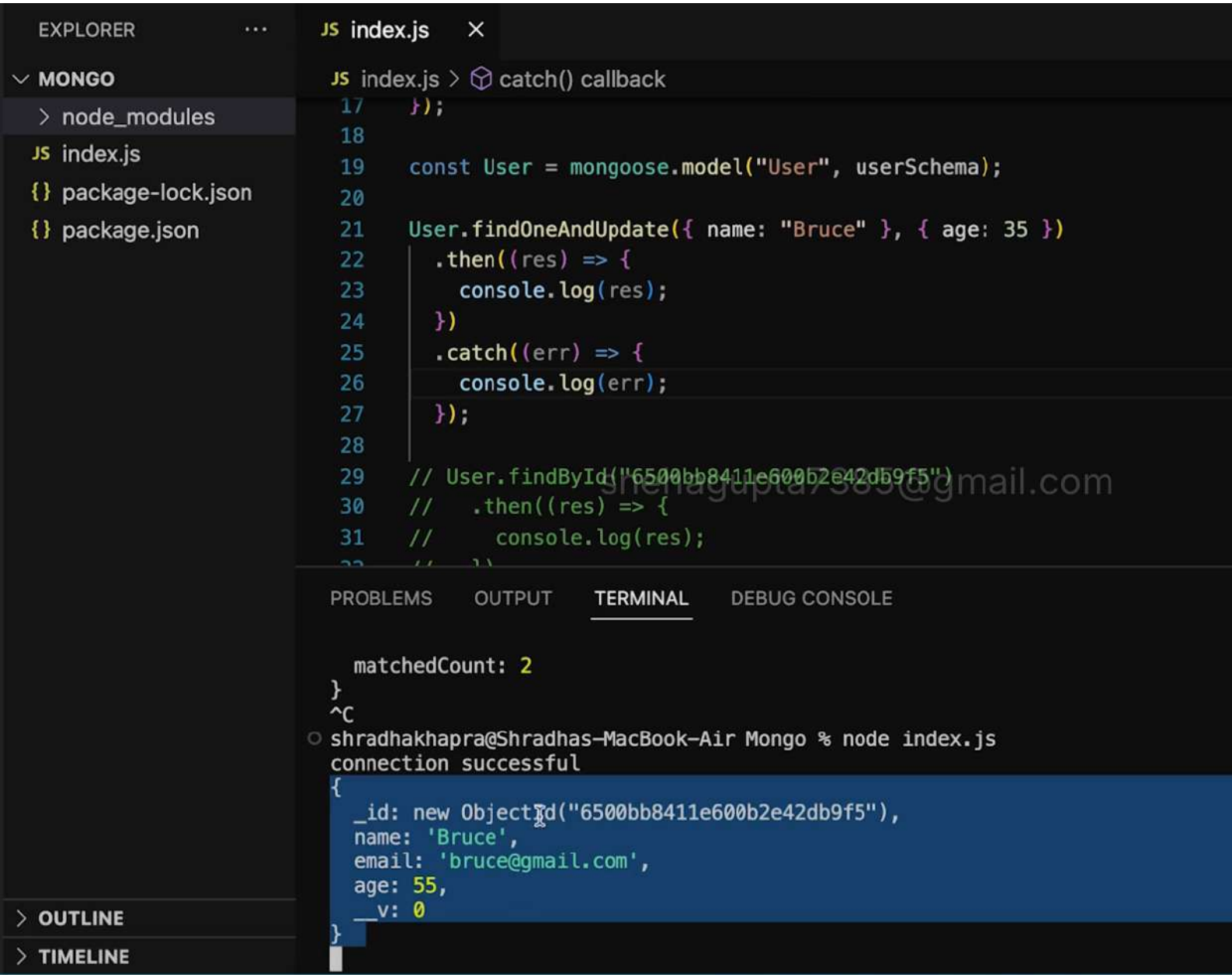
```
User.findOneAndUpdate({ name: "Tony" }, { age: 60 }, { new: true }).then(
  (data) => {
    console.log(data);
  }
);
```

Model.findByIdAndUpdate()

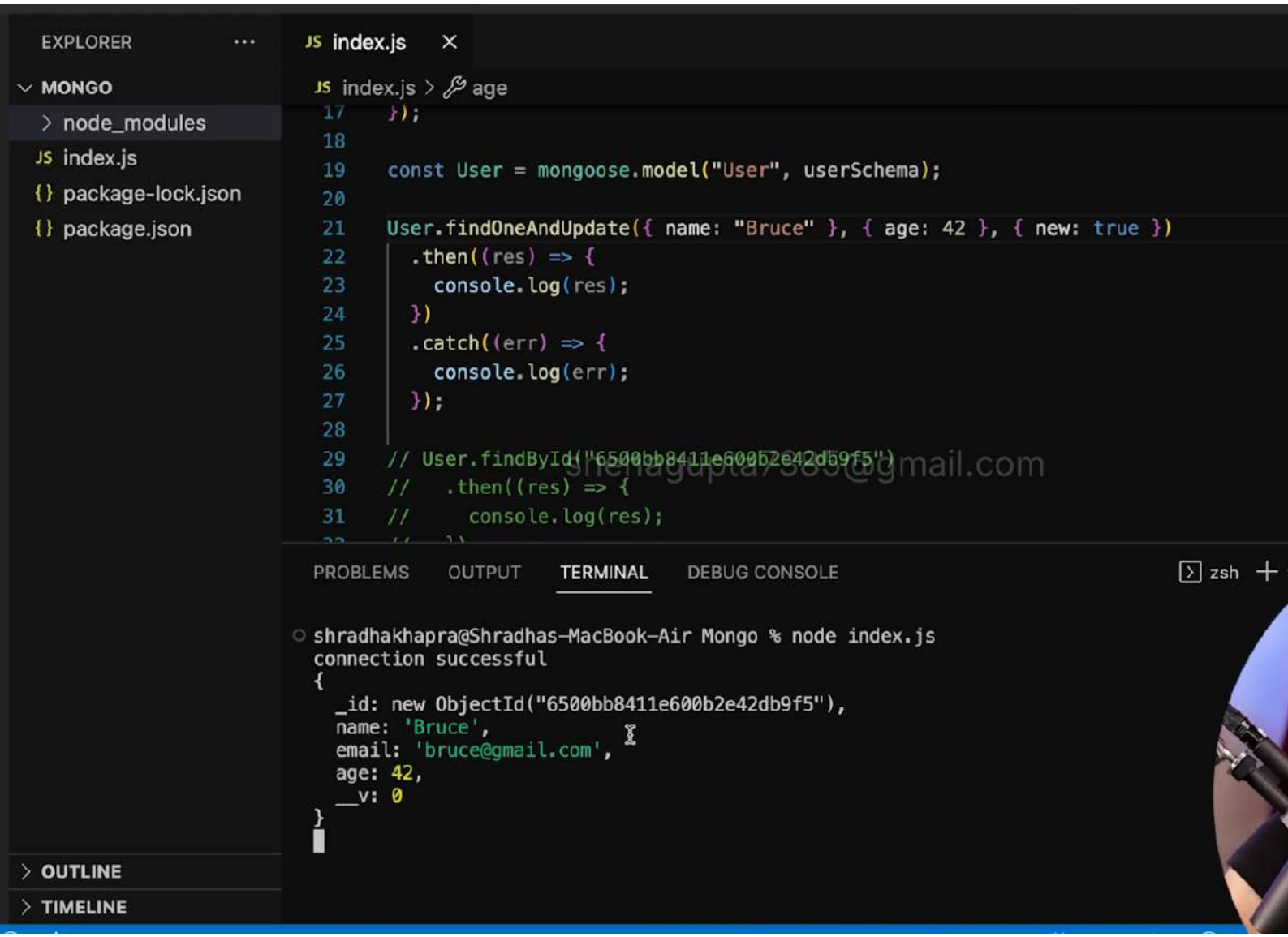
`Model.findOneAndUpdate()`

Parameters:

- `[conditions]` «Object»
- `[update]` «Object»
- `[options]` «Object» optional see `Query.prototype.setOptions()`
  - `[options.returnDocument='before']` «String» Has two possible values, `'before'` and `'after'`. By default, it will return the document before the update was applied.
  - `[options.lean]` «Object» if truthy, mongoose will return the document as a plain JavaScript object rather than a mongoose document. See `Query.lean()` and the Mongoose lean tutorial.
  - `[options.session=null]` «ClientSession» The session associated with this query. See transactions docs.
  - `[options.strict]` «Boolean|String» overwrites the schema's strict mode option
  - `[options.timestamps=null]` «Boolean» If set to `false` and schema-level timestamps are enabled, skip timestamps for this update. Note that this allows you to overwrite timestamps. Does nothing if schema-level timestamps are not set.
  - `[options.overwrite=false]` «Boolean» By default, if you don't include any update operators in `update`, Mongoose will wrap `update` in `$set` for you. This prevents you from accidentally overwriting the document. This option tells Mongoose to skip adding `$set`. An alternative to this would be using Model.findOneAndReplace(conditions, update, options, callback).
  - `[options.upsert=false]` «Boolean» if true, and no documents found, insert a new document
  - `[options.projection=null]` «Object|String|Array[String]» optional fields to return, see `Query.prototype.select()`
  - `[options.new=false]` «Boolean» if true, return the modified document rather than the original
  - `[options.fields]` «Object|String» Field selection. Equivalent to `.select(fields).findOneAndUpdate()`
  - `[options.maxTimeMS]` «Number» puts a time limit on the query - requires mongodb >= 2.6.0
  - `[options.sort]` «Object|String» if multiple docs are found by the conditions, sets the sort order


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > catch() callback
});
const User = mongoose.model("User", userSchema);
User.findOneAndUpdate({ name: "Bruce" }, { age: 35 })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});
// User.findById("6500bb8411e600b2e42db9f5")
// .then((res) => {
// console.log(res);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
matchedCount: 2
}
^C
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
email: 'bruce@gmail.com',
age: 55,
__v: 0
}
OUTLINE
TIMELINE

```
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 55,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter',
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f5"),
    name: 'Bruce',
    email: 'bruce@gmail.com',
    age: 35,
    __v: 0
  }
]
test>
```


EXPLORER
index.js
MONGO
node_modules
index.js
package-lock.json
package.json
index.js > age
});
const User = mongoose.model("User", userSchema);
User.findOneAndUpdate({ name: "Bruce" }, { age: 42 }, { new: true })
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});
// User.findById("6500bb8411e600b2e42db9f5")
// .then((res) => {
// console.log(res);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
zsh
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500bb8411e600b2e42db9f5"),
name: 'Bruce',
email: 'bruce@gmail.com',
age: 42,
__v: 0
}
OUTLINE
TIMELINE

DELETE

Model.deleteOne() *//returns count*

```
User.deleteOne({ name: "Adam" }).then((res) => {
  console.log(res);
});
```

Model.deleteMany() *//returns count*

```
User.deleteMany({ age: { $gt: 40 } }).then((res) => {
  console.log(res);
});
```

`Model.deleteOne()`

Parameters:

- `conditions` «Object»
- `[options]` «Object» optional see `Query.prototype.setOptions()`
  - `[options.translateAliases=null]` «Boolean» If set to `true`, translates any schema-defined aliases in `filter`, `projection`, `update`, and `distinct`. Throws an error if there are any conflicts where both alias and raw property are defined on the same object.

Returns:

- «Query»

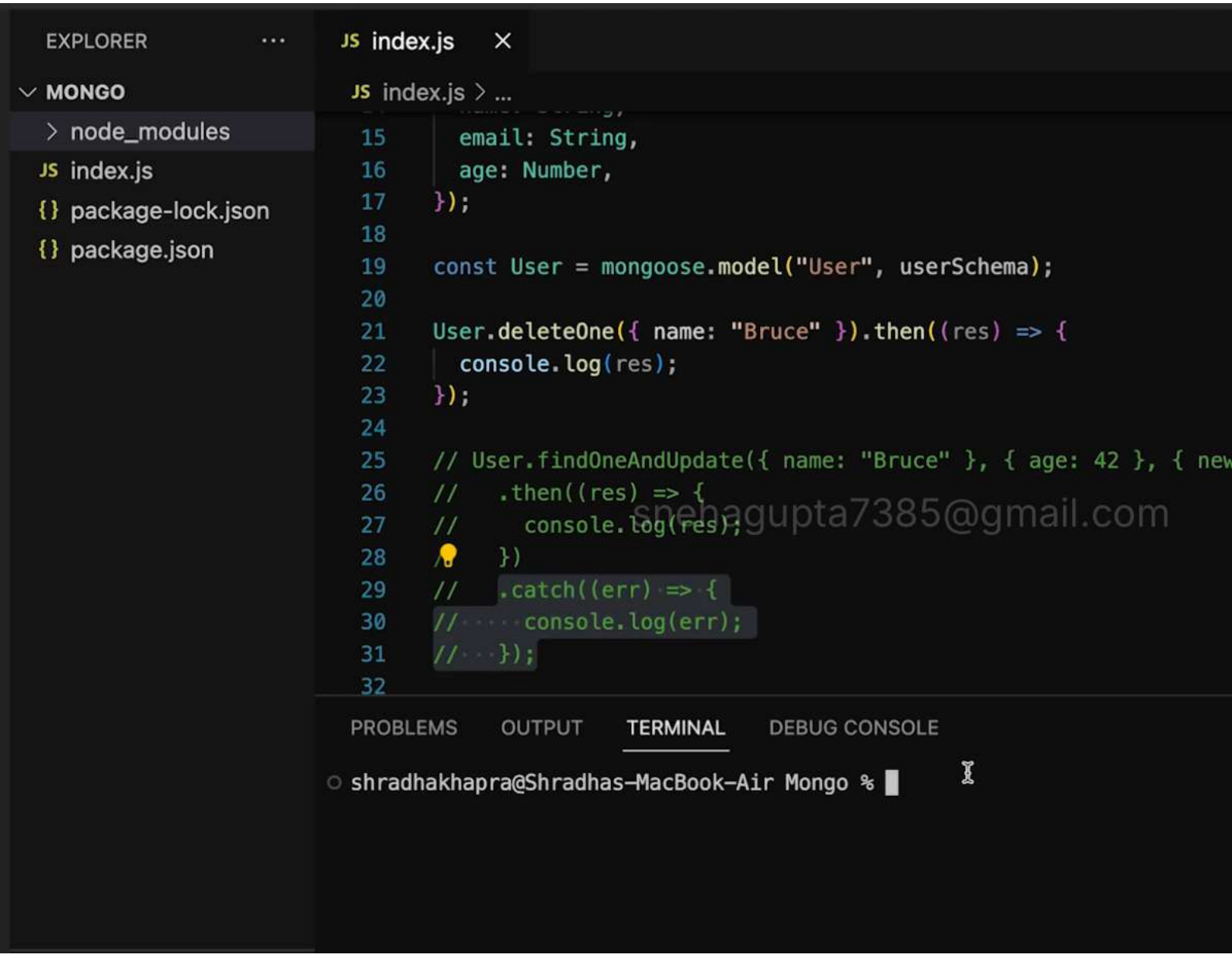
Deletes the first document that matches `conditions` from the collection. It returns an object with the property `deletedCount` indicating how many documents were deleted. Behaves like `remove()`, but deletes at most one document regardless of the `single` option.

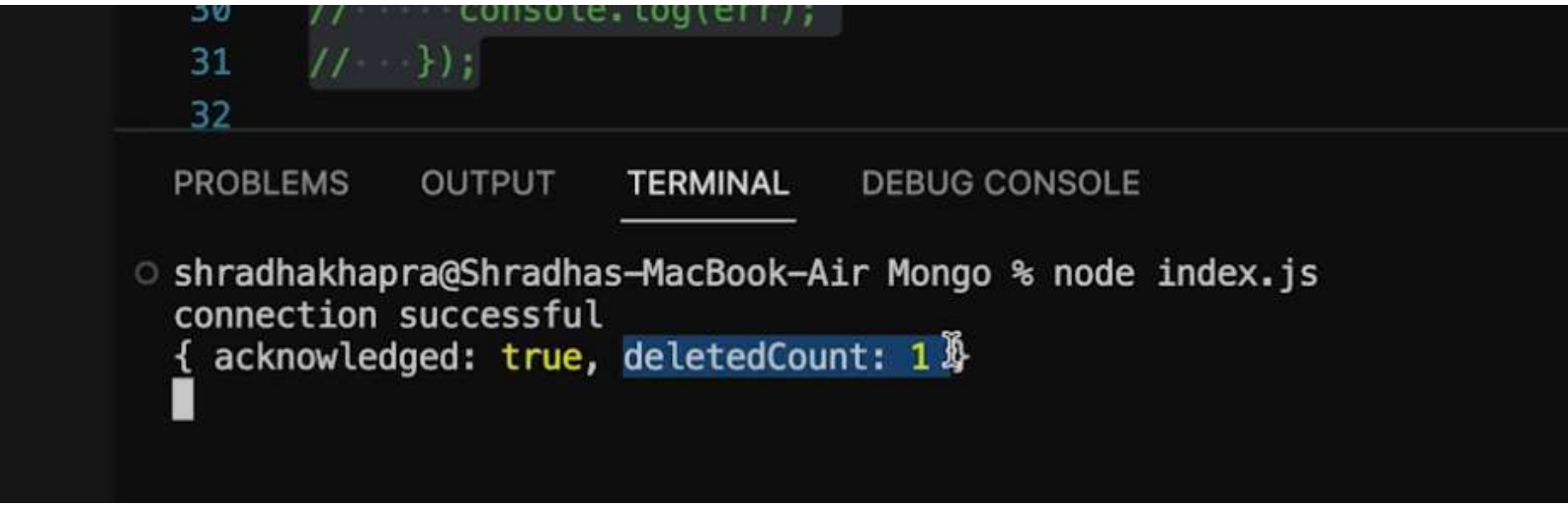
Example:

```
await Character.deleteOne({ name: 'Eddard Stark' }); // returns {deletedCount: 1}
```

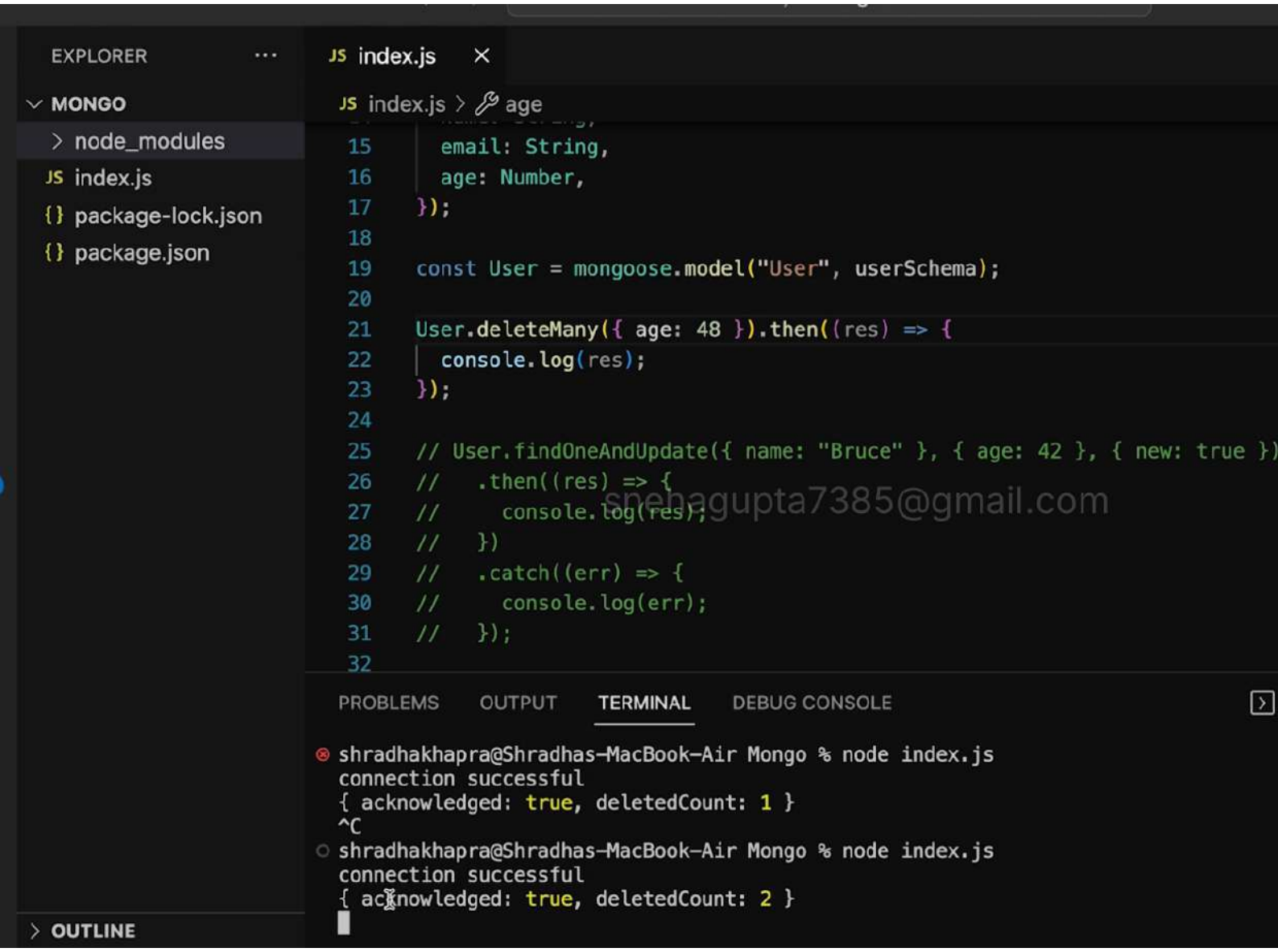
Note:

This function triggers `deleteOne` query hooks. Read the middleware docs to learn more.




//····console.log(err);
//···});
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{ acknowledged: true, deletedCount: 1 }

```
    age: 48,
    __v: 0
  },
  {
    _id: ObjectId("6500ba8a8ecc3e6eb6166393"),
    name: 'Eve',
    email: 'eve@yahoo.in',
    age: 48,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 55,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter'
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  }
]
test>
```



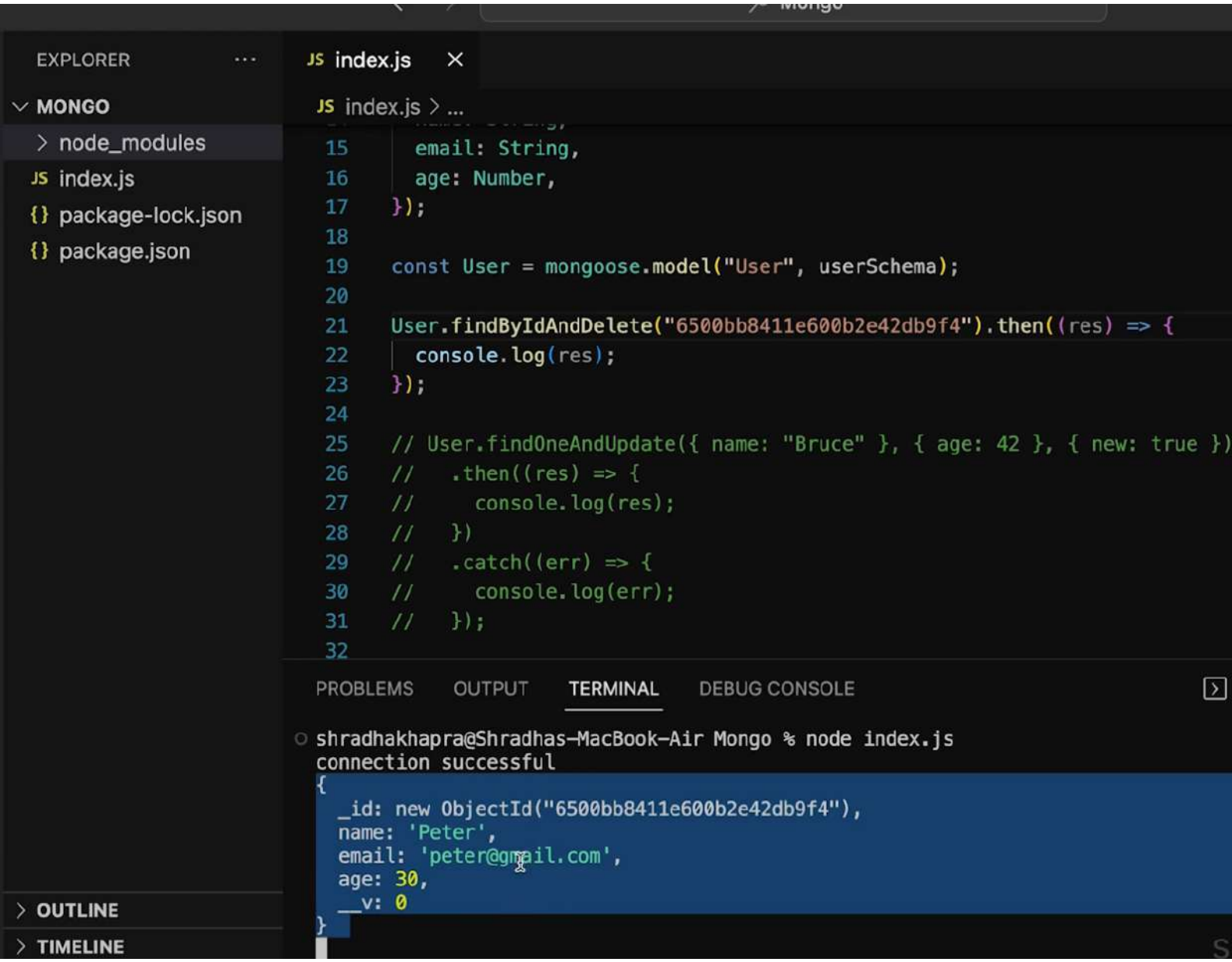
```
test> db.users.find()
[
  {
    _id: ObjectId("6500bb8411e600b2e42db9f3"),
    name: 'Tony',
    email: 'tony@gmail.com',
    age: 55,
    __v: 0
  },
  {
    _id: ObjectId("6500bb8411e600b2e42db9f4"),
    name: 'Peter',
    email: 'peter@gmail.com',
    age: 30,
    __v: 0
  }
]
test>
```

DELETE

Try Yourself

Model.findByIdAndDelete( )

Model.findOneAndDelete( )


EXPLORER
MONGO
node_modules
index.js
package-lock.json
package.json
OUTLINE
TIMELINE
index.js
index.js > ...
email: String,
age: Number,
});
const User = mongoose.model("User", userSchema);
User.findByIdAndDelete("6500bb8411e600b2e42db9f4").then((res) => {
console.log(res);
});
// User.findOneAndUpdate({ name: "Bruce" }, { age: 42 }, { new: true })
// .then((res) => {
// console.log(res);
// })
// .catch((err) => {
// console.log(err);
// });
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node index.js
connection successful
{
_id: new ObjectId("6500bb8411e600b2e42db9f4"),
name: 'Peter',
email: 'peter@gmail.com',
age: 30,
__v: 0
}


}
]
test> db.users.find()
[
{
_id: ObjectId("6500bb8411e600b2e42db9f3"),
name: 'Tony',
email: 'tony@gmail.com',
age: 55,
__v: 0
}
]
test>

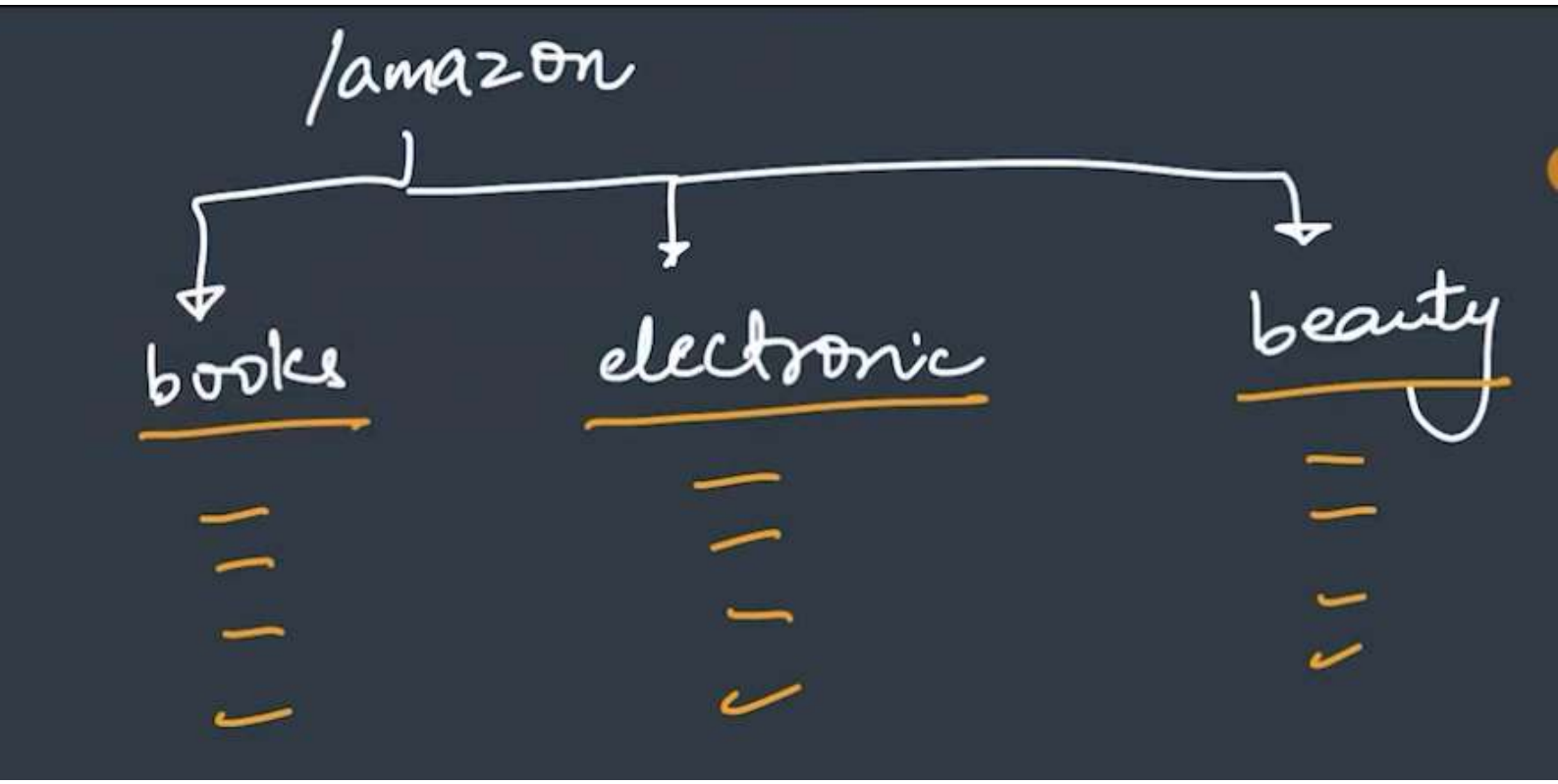
Schema Validations

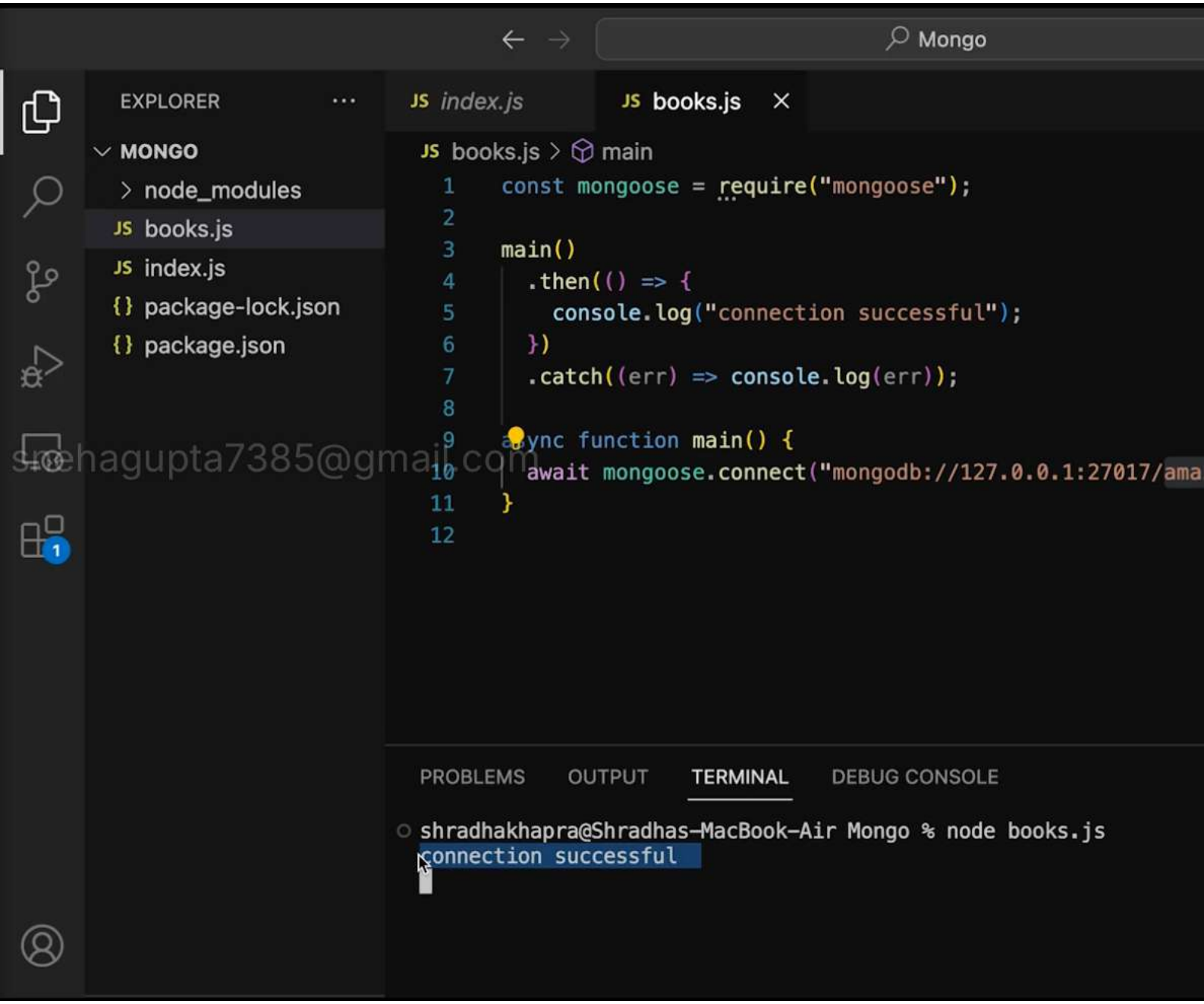
Basically, Rules for Schema

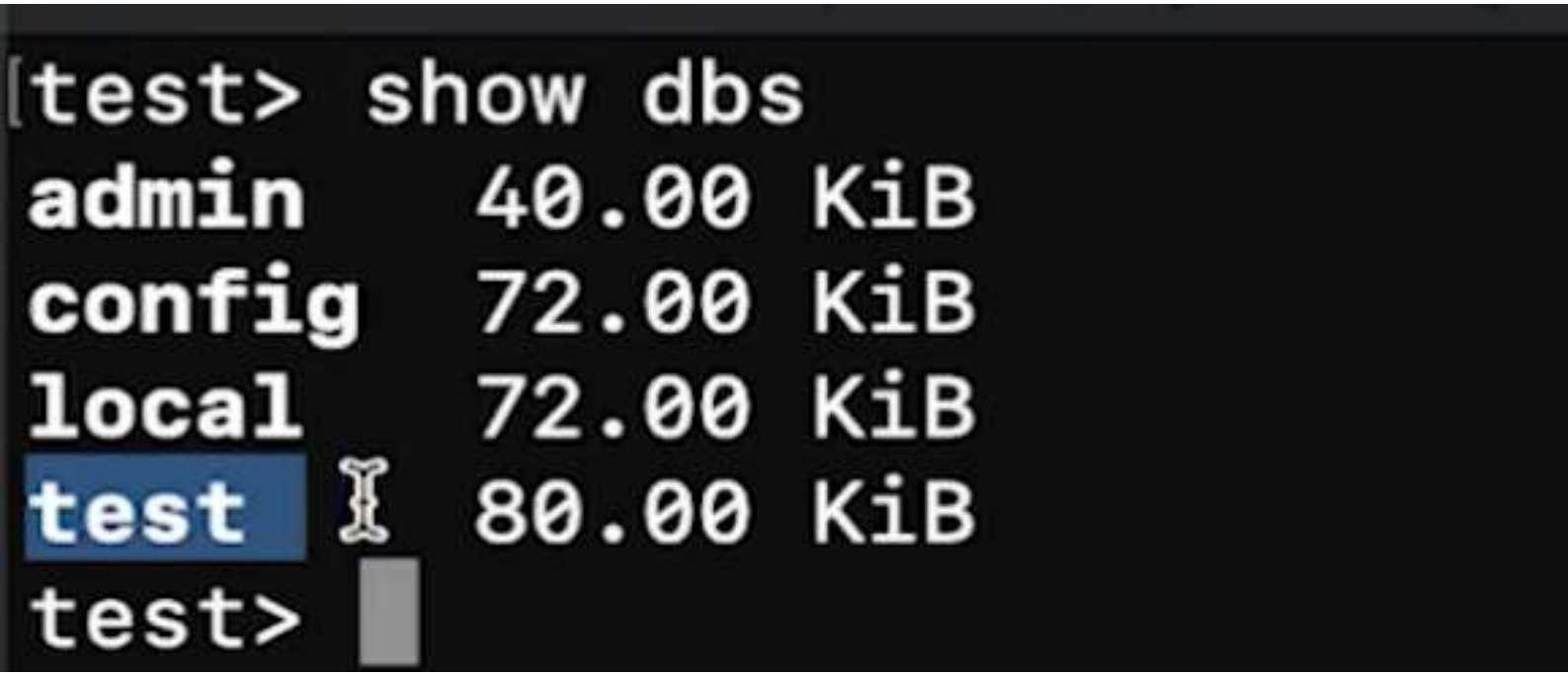
```
const bookSchema = mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  author: {
    type: String,
  },
  price: {
    type: Number,
  },
});
```

SQL → cols → name, datatype, constraints

schema


/amazon
books
electronic
beauty


Mongo
EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
index.js
books.js
books.js > main
1 const mongoose = require("mongoose");
2
3 main()
4 .then(() => {
5 console.log("connection successful");
6 })
7 .catch((err) => console.log(err));
8
9 async function main() {
10 await mongoose.connect("mongodb://127.0.0.1:27017/ama
11 }
12
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful


test> show dbs
admin 40.00 KiB
config 72.00 KiB
local 72.00 KiB
test 80.00 KiB
test>

```
m.save(callback);
```

The `type` Key

`type` is a special property in Mongoose schemas. When Mongoose finds a nested property named `type` in your schema, Mongoose assumes that it needs to define a SchemaType with the given type.

```
// 3 string SchemaTypes: 'name', 'nested.firstName', 'nested.lastName'
const schema = new Schema({
  name: { type: String },
  nested: {
    firstName: { type: String },
    lastName: { type: String }
  }
});
```

As a consequence, you need a little extra work to define a property named `type` in your schema. For example, suppose you're building a stock portfolio app, and you want to store the asset's `type` (stock, bond, ETF, etc.). Naively, you might define your schema as shown below:

```
const holdingSchema = new Schema({
  // You might expect `asset` to be an object that has 2 properties,
  // but unfortunately `type` is special in Mongoose so mongoose
  // interprets this schema to mean that `asset` is a string
  asset: {
    type: String,
    ticker: String
  }
});
```

However, when Mongoose sees `type: String`, it assumes that you mean `asset` should be a string, not an

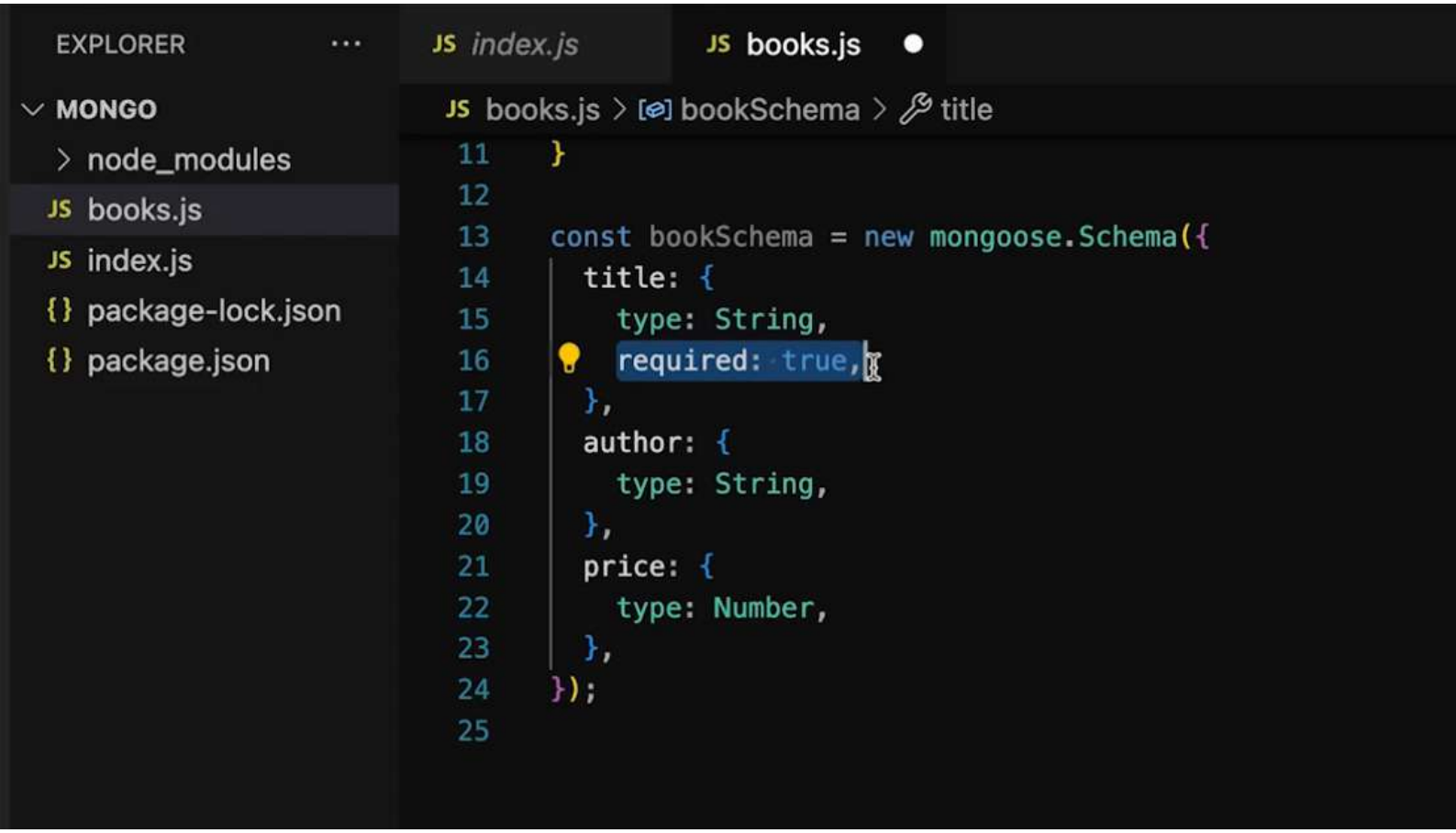
`true` in your SchemaType options. Mongoose comes with support for several built-in SchemaType options, like `lowercase` in the above example.

The `lowercase` option only works for strings. There are certain options which apply for all schema types, and some that apply for specific schema types.

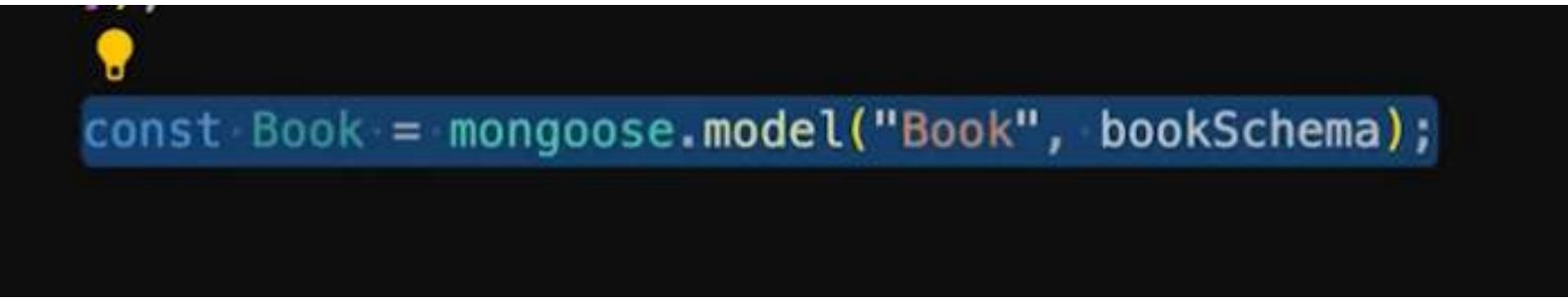
All Schema Types

- `required` : boolean or function, if true adds a required validator for this property
- `default` : Any or function, sets a default value for the path. If the value is a function, the return value of the function is used as the default.
- `select` : boolean, specifies default projections for queries
- `validate` : function, adds a validator function for this property
- `get` : function, defines a custom getter for this property using `Object.defineProperty()` .
- `set` : function, defines a custom setter for this property using `Object.defineProperty()` .
- `alias` : string, mongoose >= 4.10.0 only. Defines a virtual with the given name that gets/sets this path.
- `immutable` : boolean, defines path as immutable. Mongoose prevents you from changing immutable paths unless the parent document has `isNew: true` .
- `transform` : function, Mongoose calls this function when you call `Document#toJSON()` function, includin when you `JSON.stringify()` a document.

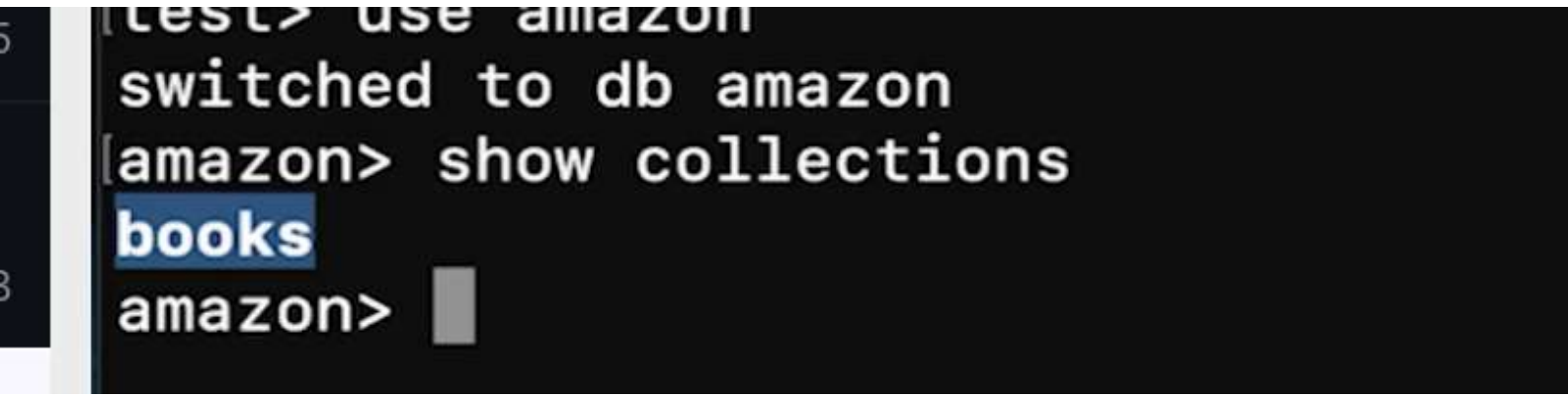
```
const numberSchema = new Schema({
```

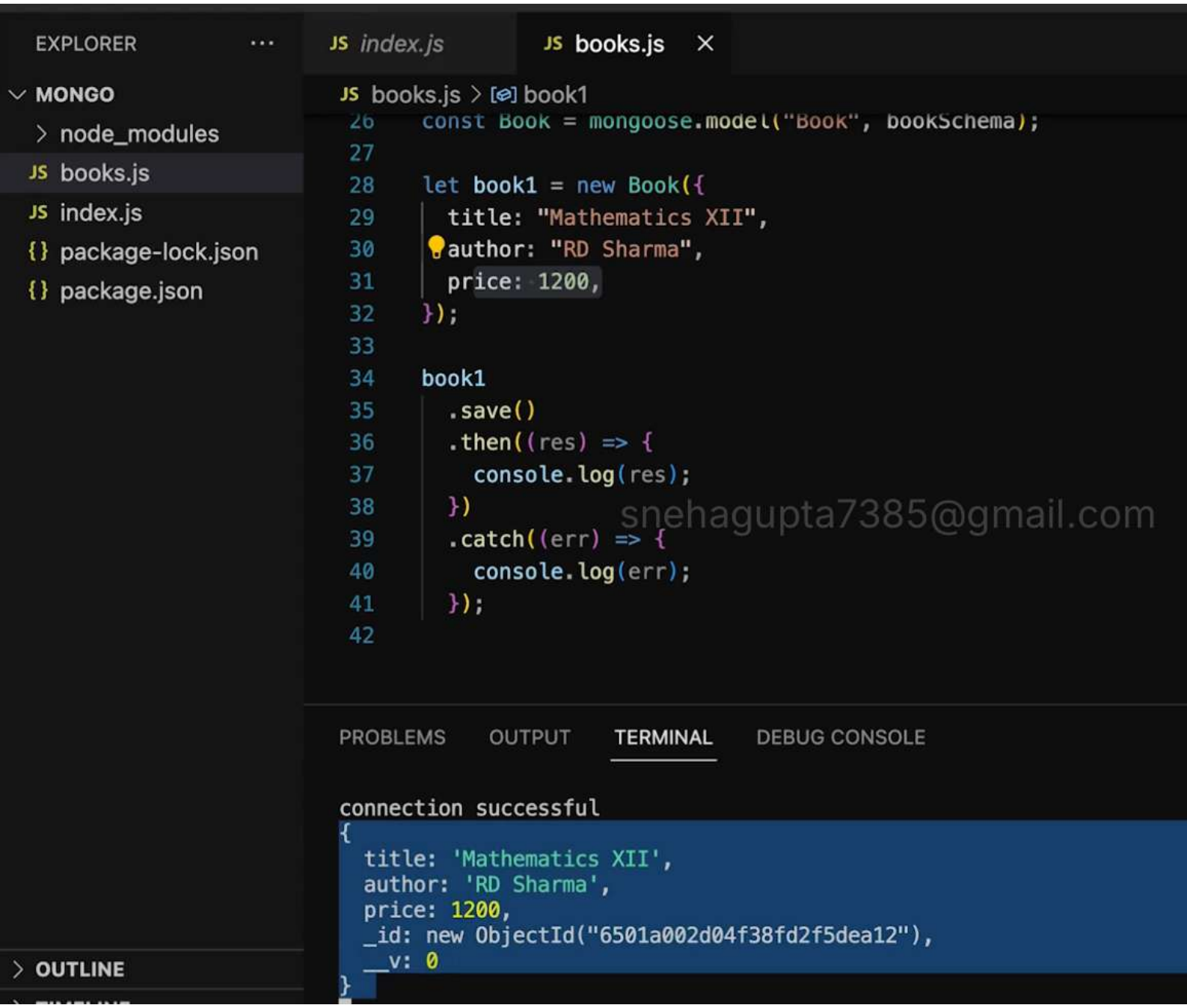

EXPLORER
index.js
books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > bookSchema > title
}
const bookSchema = new mongoose.Schema({
title: {
type: String,
required: true,
},
author: {
type: String,
},
price: {
type: Number,
},
});

```
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful
```


const Book = mongoose.model("Book", bookSchema);


test 80.00 KiB
test> show dbs
admin 40.00 KiB
amazon 8.00 KiB
config 96.00 KiB
local 72.00 KiB
test 80.00 KiB
test> use amazon
switched to db amazon


test> use amazon
switched to db amazon
amazon> show collections
books
amazon>



```
amazon> db.books.find()
[
  {
    _id: ObjectId("6501a002d04f38fd2f5dea12"),
    title: 'Mathematics XII',
    author: 'RD Sharma',
    price: 1200,
    __v: 0
  }
]
amazon>
```

JS books.js > bookSchema > title

```
24    });
25
26    const Book = mongoose.model("Book", bookSchema);
27
28    let book1 = new Book({
29      author: "RD Sharma",
30      price: 1200,
31    });
32
33    book1
34      .save()
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE


```
  errors: {
    title: ValidatorError: Path `title` is required.
        at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mon
e.js:1365:13)
        at SchemaString.SchemaType.doValidate (/Users/shradhakhapra/
les/mongoose/lib/schematype.js:1349:7)
        at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_mo
        at processTicksAndRejections (node:internal/process/task_que
      properties: [Object],
      kind: 'required',
      path: 'title',
      value: undefined,
      reason: undefined,
      [Symbol(mongoose:validatorError)]: true
    }
  },
  _message: 'Book validation failed'
}
connection successful
```

JS index.js JS books.js ×

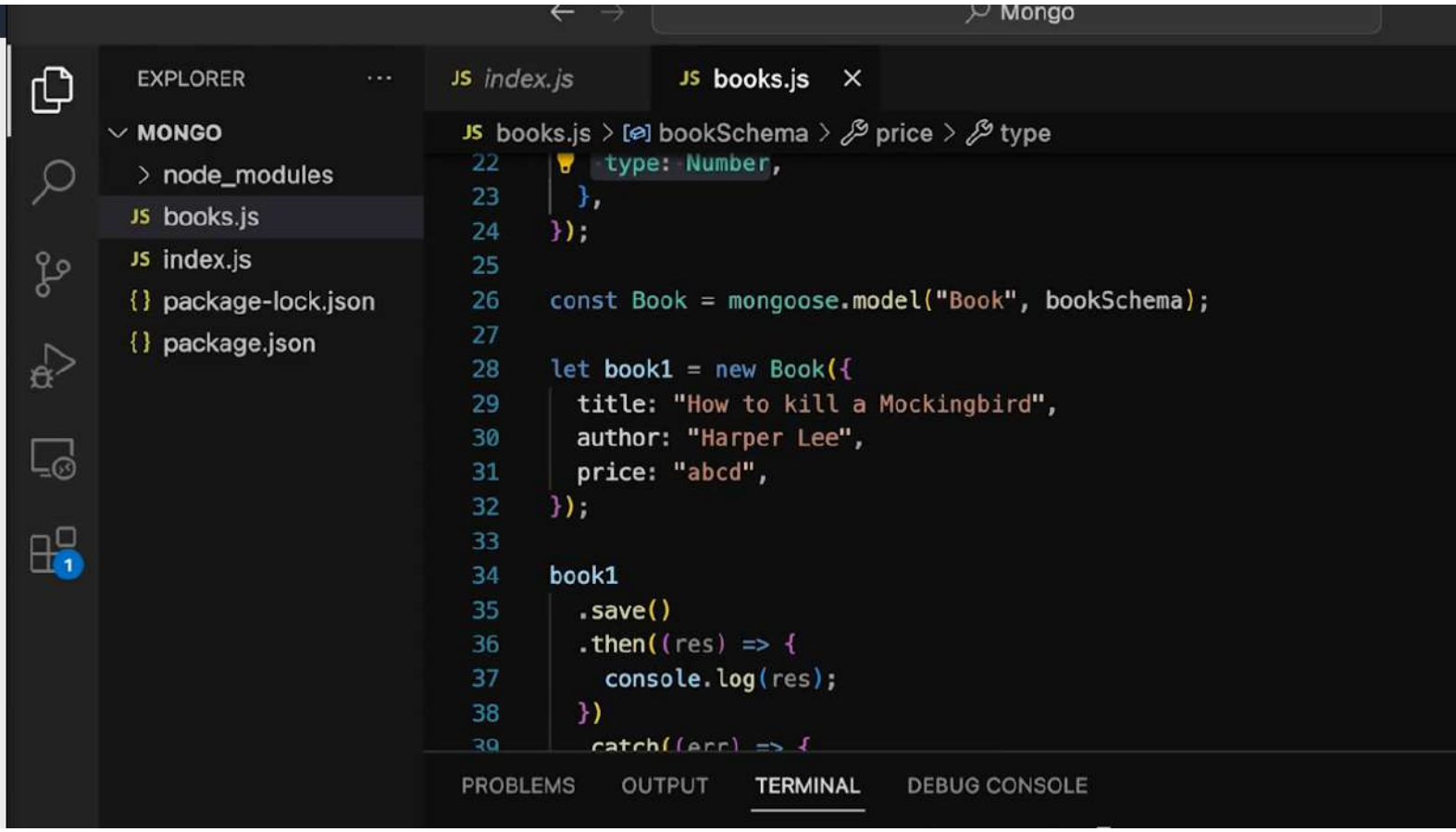
JS books.js > [] bookSchema

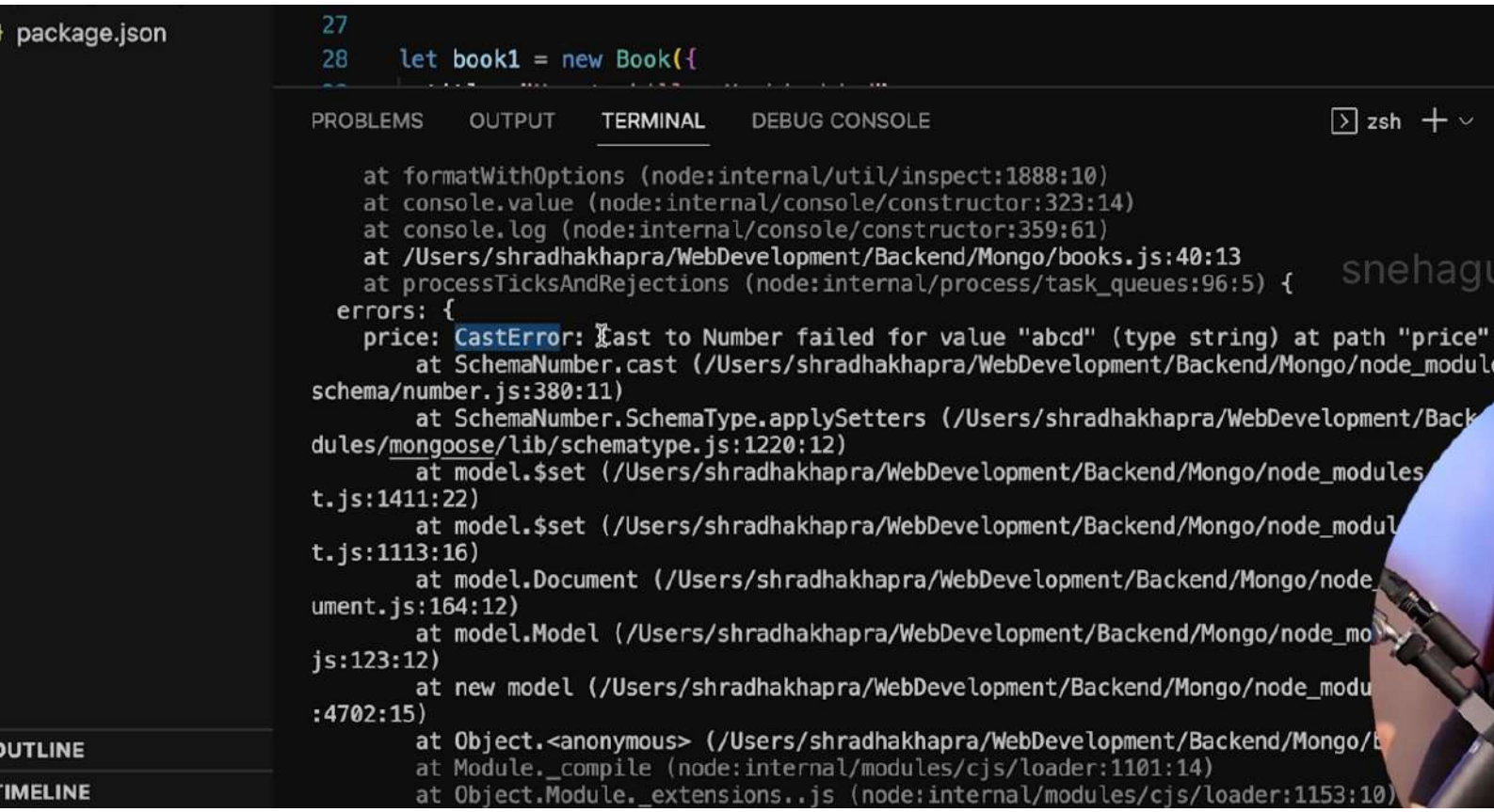
```
13  const bookSchema = new mongoose.Schema({
14    title: {
15      type: String,
16      required: true,
17    },
18    author: {
19      type: String,
20    },
21    price: {
22      type: Number,
23    },
24  });
25
26  const Book = mongoose.model("Book", bookSchema);
27
28  let book1 = new Book({
29    title: "Mathematics VIII",
```

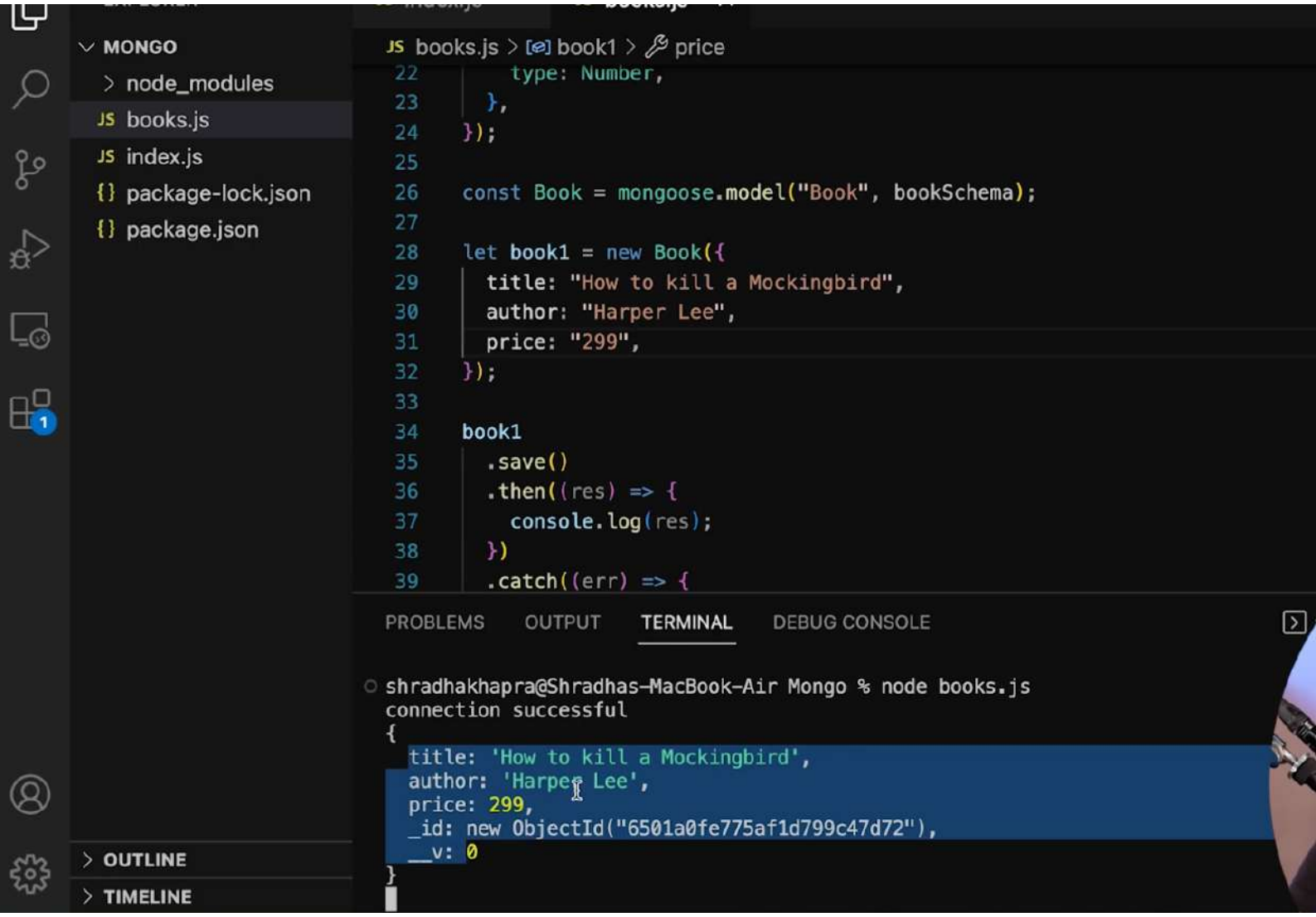
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful
{
  title: 'Mathematics VIII',
  price: 1200,
  _id: new ObjectId("6501a069960bcd9dfeaa7ea1"),
  __v: 0
}
```

```
    __v: 0
  }
]
amazon> db.books.find()
[
  {
    _id: ObjectId("6501a002d04f38fd2f5dea12"),
    title: 'Mathematics XII',
    author: 'RD Sharma',
    price: 1200,
    __v: 0
  },
  {
    _id: ObjectId("6501a069960bcd9dfeaa7ea1"),
    title: 'Mathematics VIII',
    price: 1200,
    __v: 0
  }
]
amazon>
```


Mongo
EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
index.js
books.js
books.js > bookSchema > price > type
type: Number,
},
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "How to kill a Mockingbird",
author: "Harper Lee",
price: "abcd",
});
book1
.save()
.then((res) => {
console.log(res);
})
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE


package.json
27
28 let book1 = new Book({
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
zsh
at formatWithOptions (node:internal/util/inspect:1888:10)
at console.value (node:internal/console/constructor:323:14)
at console.log (node:internal/console/constructor:359:61)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:40:13
at processTicksAndRejections (node:internal/process/task_queues:96:5) {
errors: {
price: CastError: Cast to Number failed for value "abcd" (type string) at path "price"
at SchemaNumber.cast (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modul
schema/number.js:380:11)
at SchemaNumber.SchemaType.applySetters (/Users/shradhakhapra/WebDevelopment/Back
dules/mongoose/lib/schematype.js:1220:12)
at model.$set (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modules
t.js:1411:22)
at model.$set (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modul
t.js:1113:16)
at model.Document (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_
ument.js:164:12)
at model.Model (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_mo
js:123:12)
at new model (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modu
:4702:15)
at Object.<anonymous> (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/b
at Module._compile (node:internal/modules/cjs/loader:1101:14)
at Object.Module._extensions..js (node:internal/modules/cjs/loader:1153:10)
OUTLINE
TIMELINE

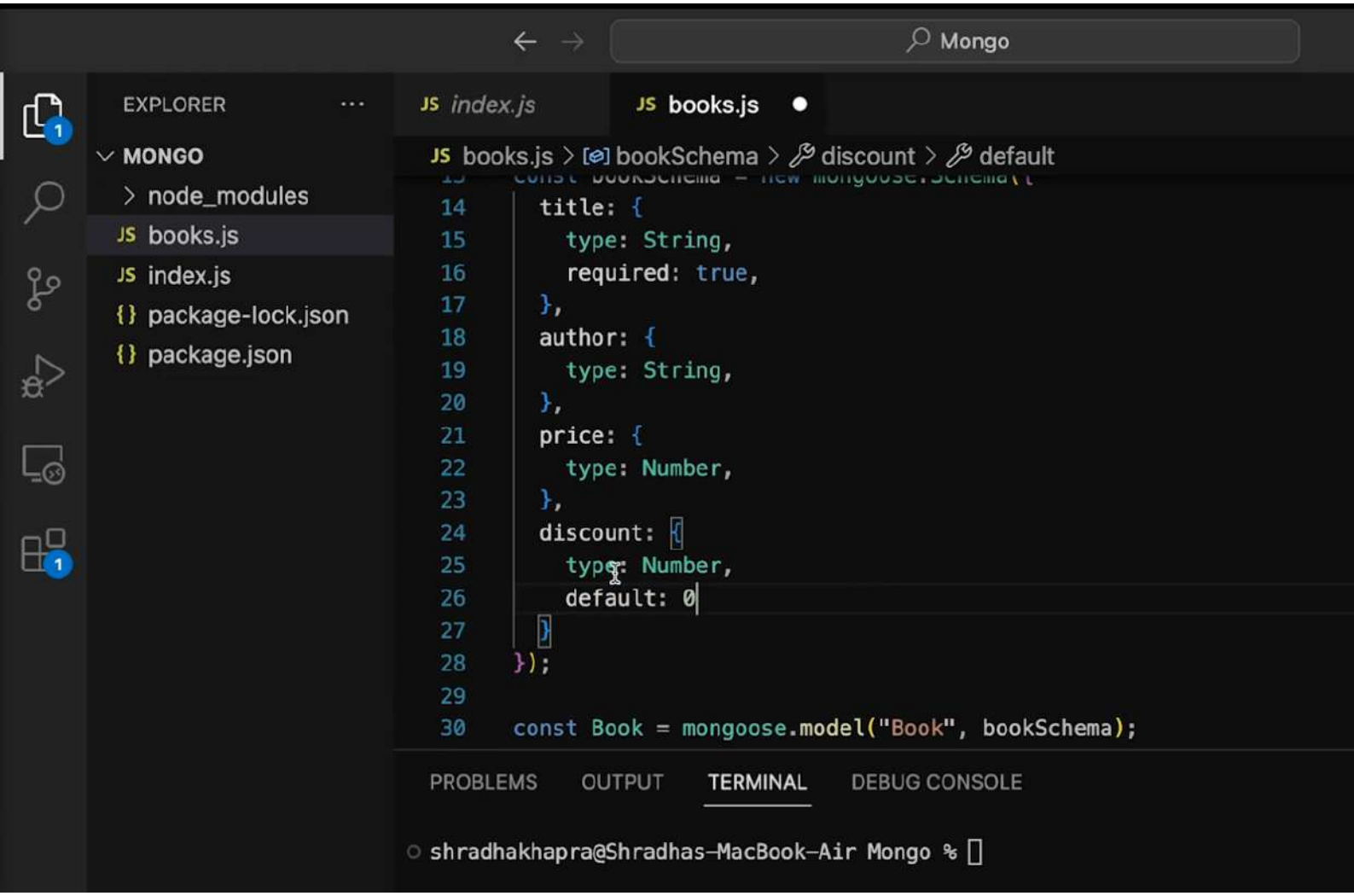

MONGO
node_modules
books.js
index.js
package-lock.json
package.json
OUTLINE
TIMELINE
books.js > book1 > price
type: Number,
},
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "How to kill a Mockingbird",
author: "Harper Lee",
price: "299",
});
book1
.save()
.then((res) => {
console.log(res);
})
.catch((err) => {
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful
{
title: 'How to kill a Mockingbird',
author: 'Harper Lee',
price: 299,
_id: new ObjectId("6501a0fe775af1d799c47d72"),
__v: 0
}

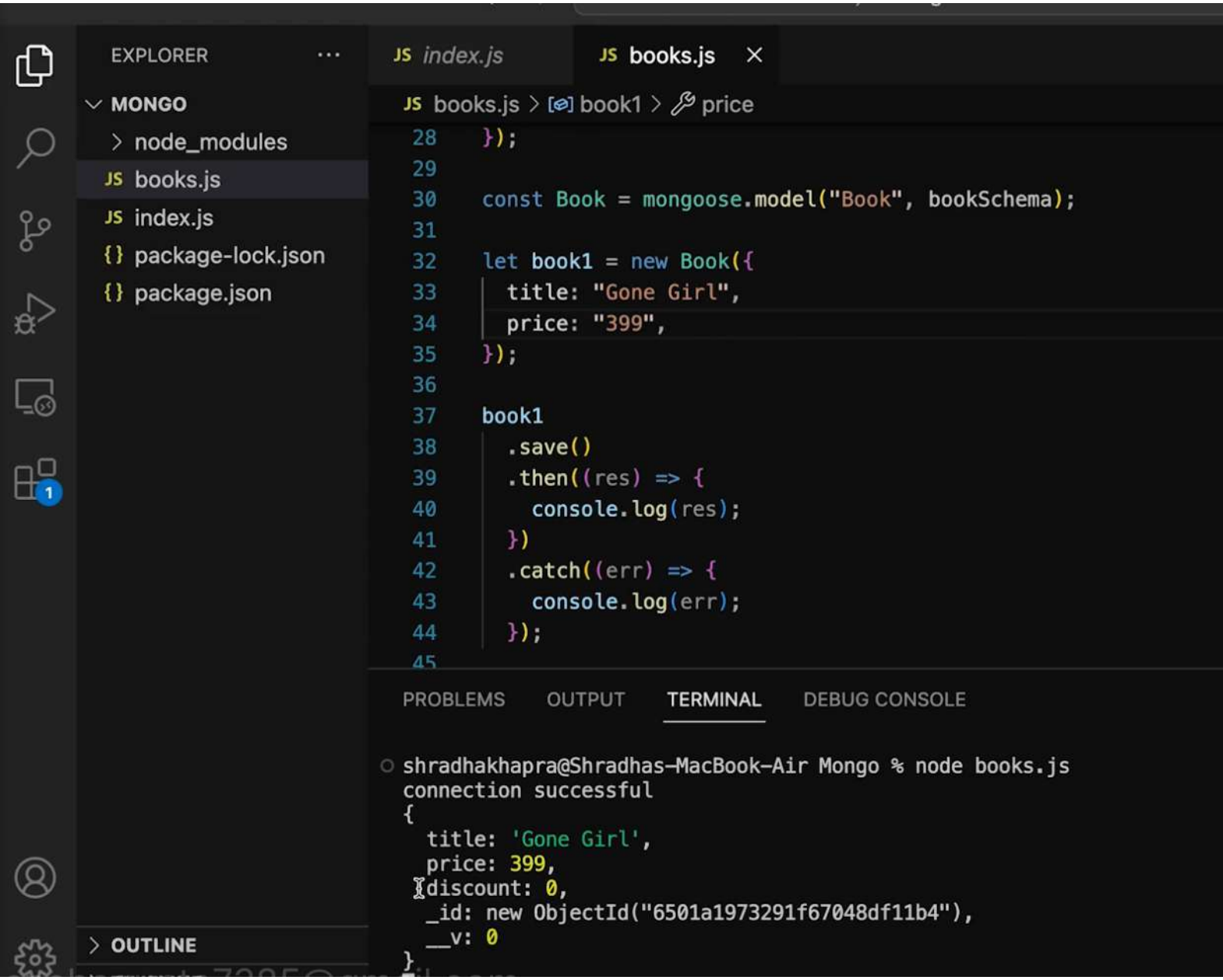

v: 0
d: ObjectId("6501a0fe775af1d799c47d72"),
tle: 'How to kill a Mockingbird',
thor: 'Harper Lee',
ice: 299,
v: 0

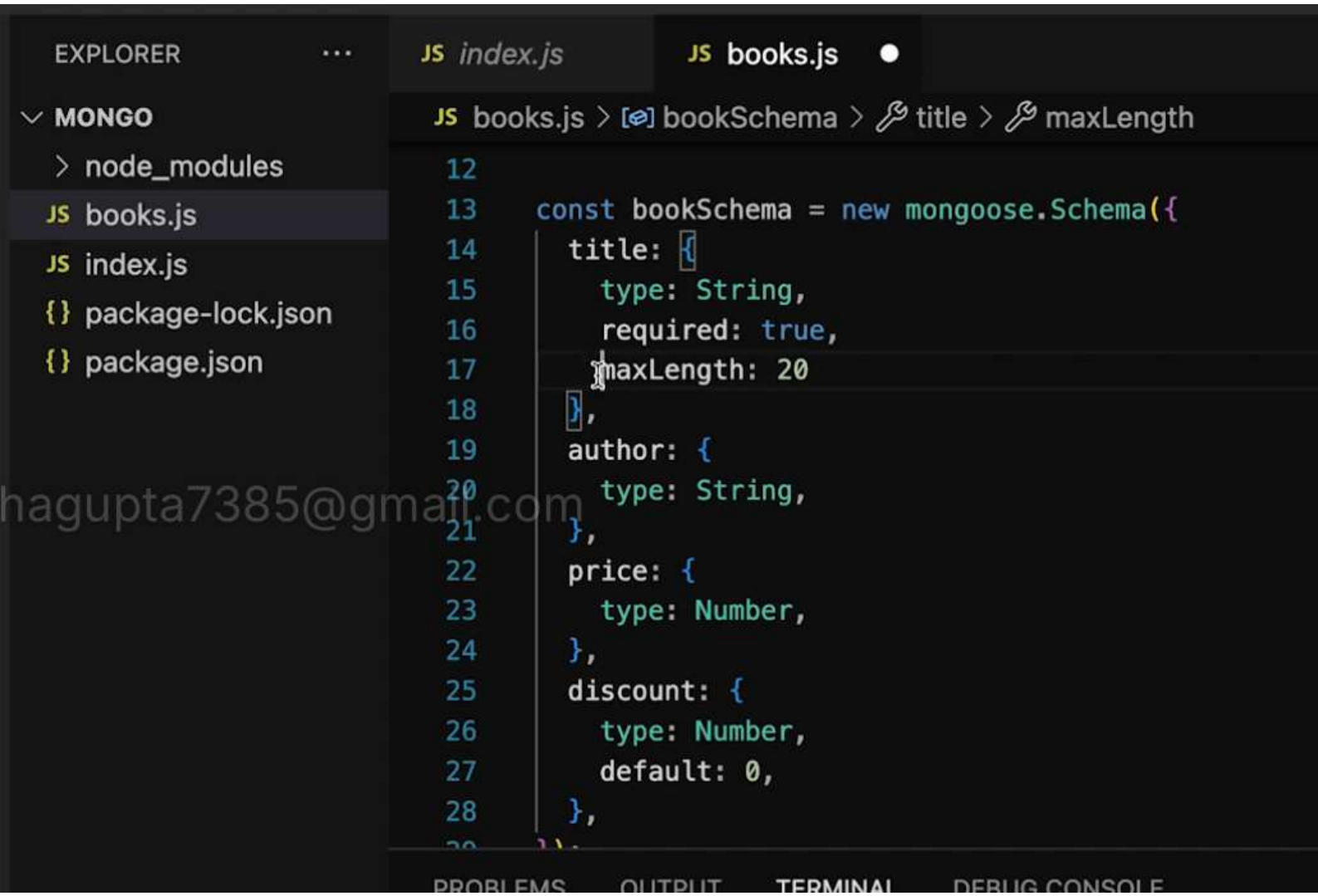
Schema Validations

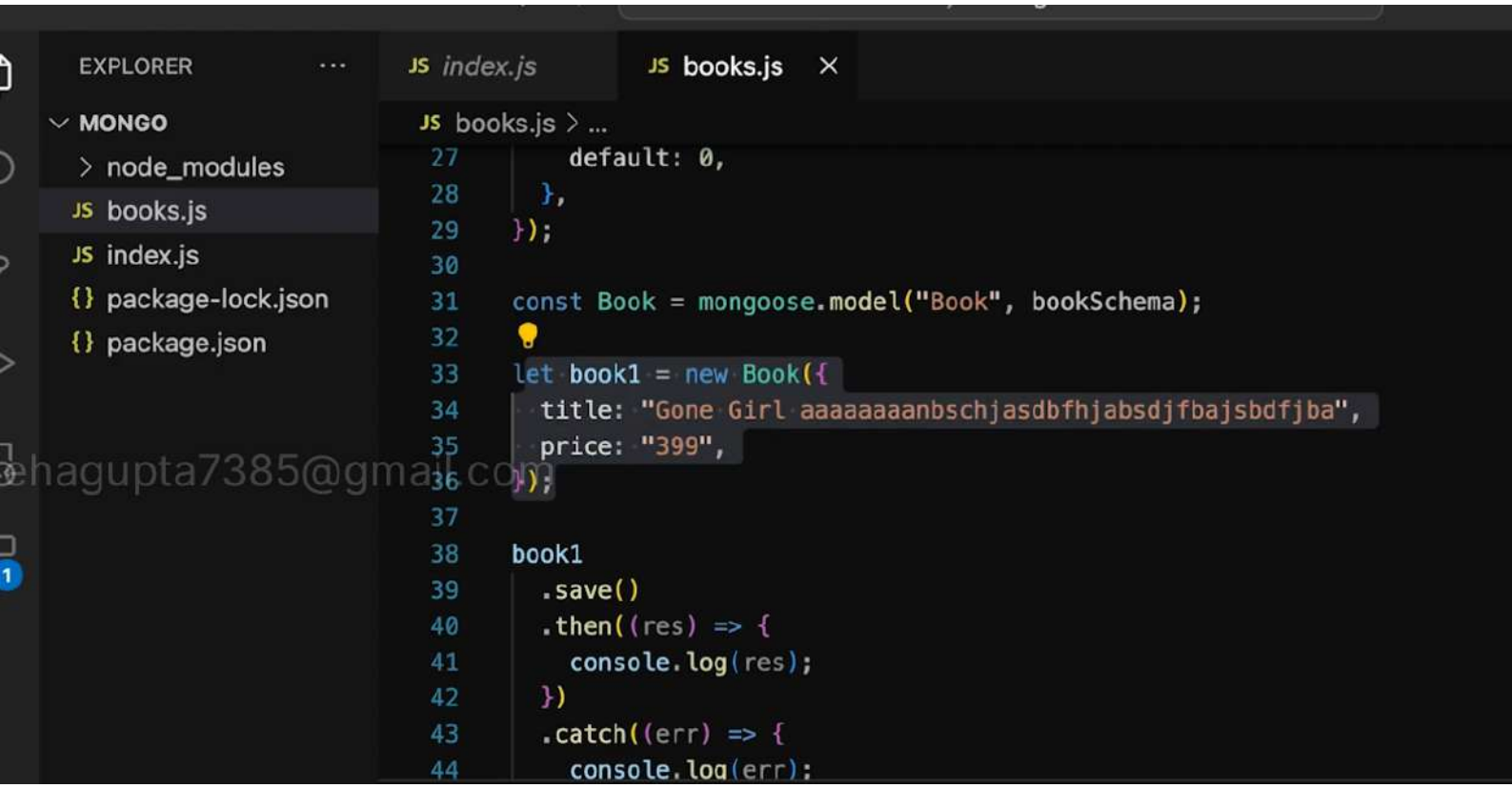
SchemaType Options

```
const bookSchema = mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  author: {
    type: String,
  },
  price: {
    type: Number,
    min: [1, "please enter a valid price"],
  },
  discount: {
    type: Number,
    default: 0,
  },
  genre: [String],
  category: {
    type: String,
    enum: ["fiction", "non-fiction"],
  },
});
```


Mongo
EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
index.js
books.js
books.js > bookSchema > discount > default
14 title: {
15 type: String,
16 required: true,
17 },
18 author: {
19 type: String,
20 },
21 price: {
22 type: Number,
23 },
24 discount: {
25 type: Number,
26 default: 0
27 }
28 });
29
30 const Book = mongoose.model("Book", bookSchema);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo %

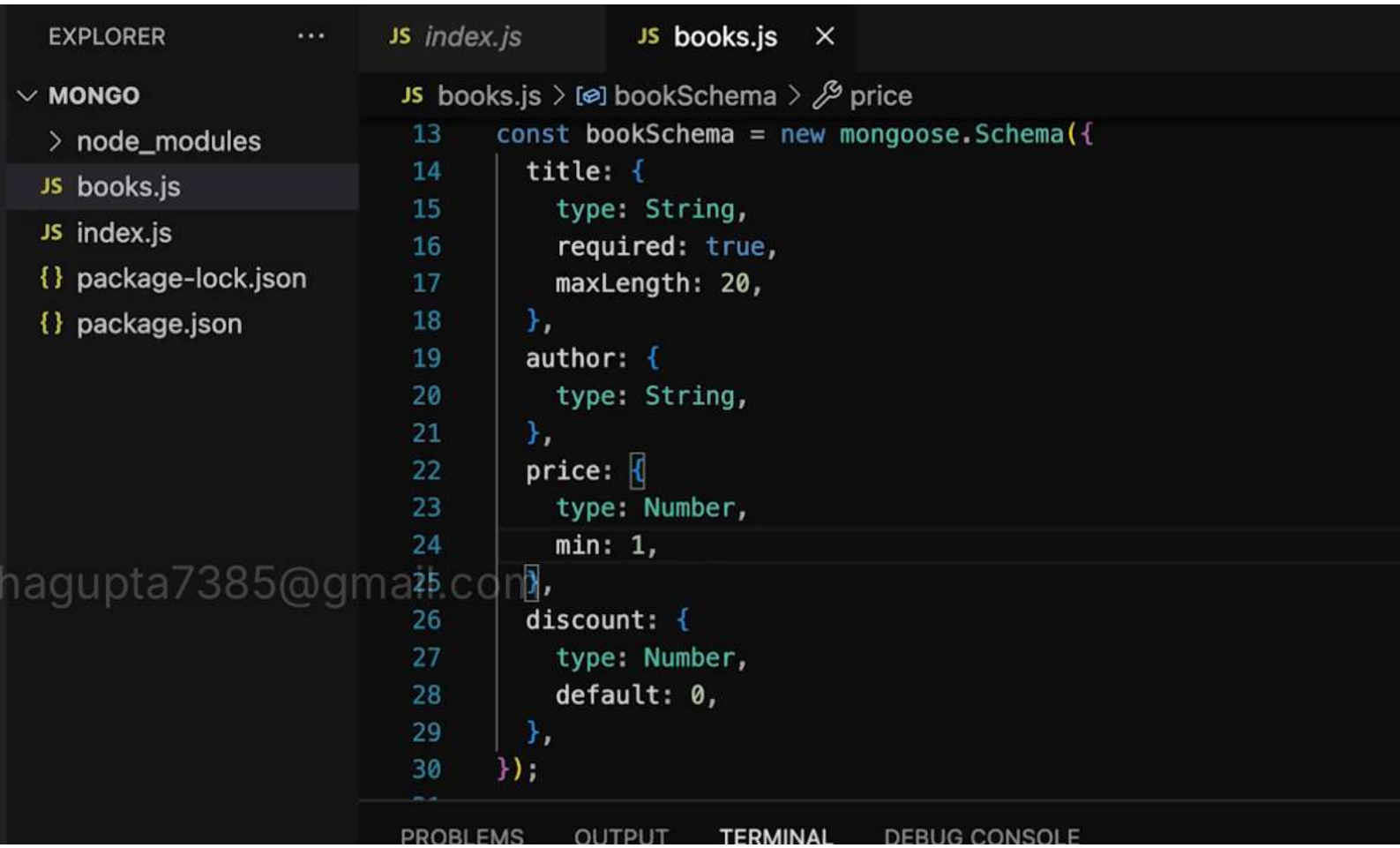

EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
OUTLINE
index.js
books.js
books.js > book1 > price
28 });
29
30 const Book = mongoose.model("Book", bookSchema);
31
32 let book1 = new Book({
33 title: "Gone Girl",
34 price: "399",
35 });
36
37 book1
38 .save()
39 .then((res) => {
40 console.log(res);
41 })
42 .catch((err) => {
43 console.log(err);
44 });
45
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful
{
title: 'Gone Girl',
price: 399,
discount: 0,
_id: new ObjectId("6501a1973291f67048df11b4"),
__v: 0
}

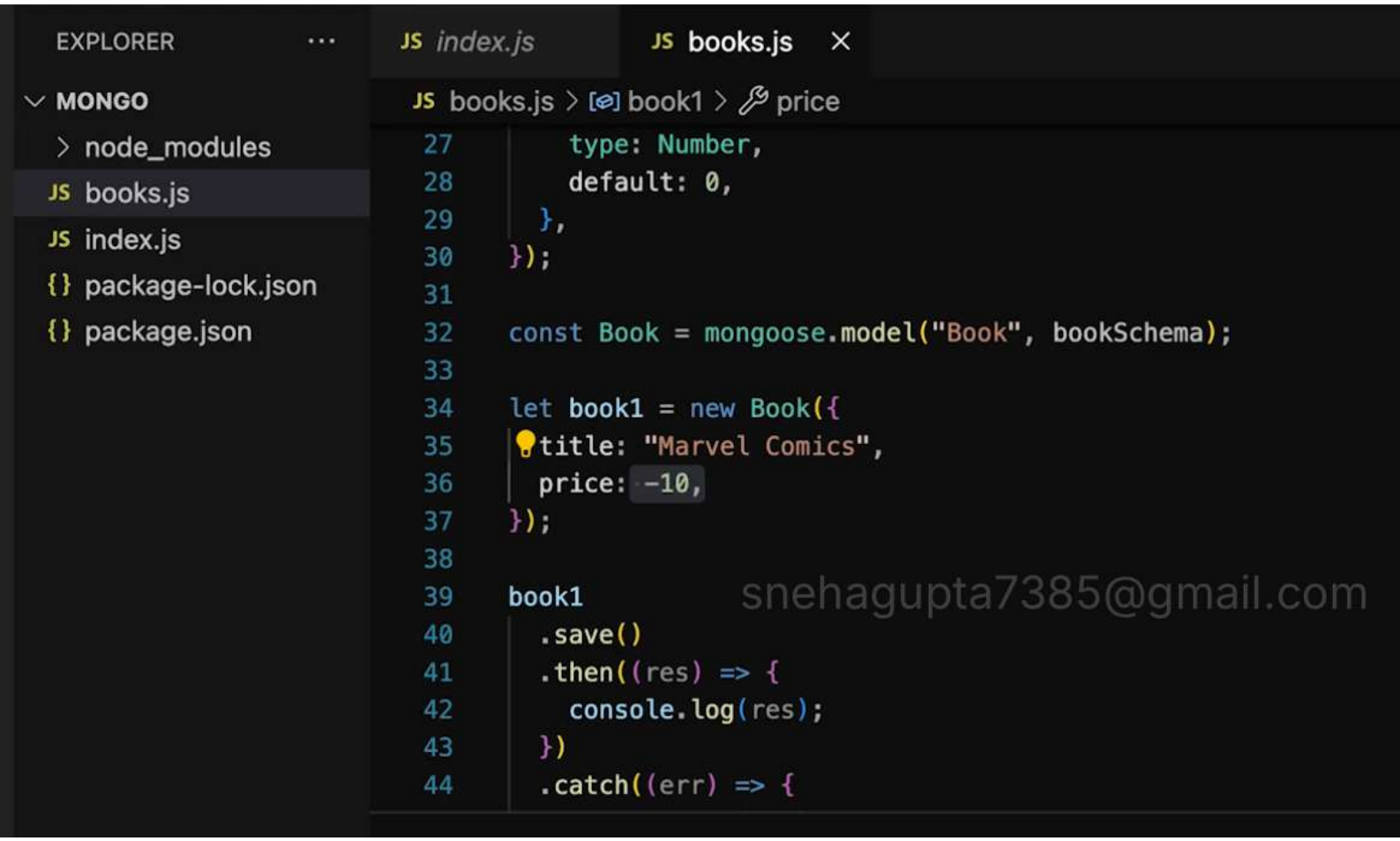




PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE zsh

```
connection successful
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
Error: Book validation failed: title: Path `title` (`Gone Girl aaaaaaaanbschjasdbfhjabsdjfbajsbdfjb
longer than the maximum allowed length (20).
    at ValidationError.inspect (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modules/mong
b/error/validation.js:50:26)
    at formatValue (node:internal/util/inspect:763:19)
    at inspect (node:internal/util/inspect:340:10)
    at formatWithOptionsInternal (node:internal/util/inspect:2006:40)
    at formatWithOptions (node:internal/util/inspect:1888:10)
    at console.value (node:internal/console/constructor:323:14)
    at console.log (node:internal/console/constructor:359:61)
    at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:44:13
    at processTicksAndRejections (node:internal/process/task_queues:96:5) {
  errors: {
    title: ValidatorError: Path `title` (`Gone Girl aaaaaaaanbschjasdbfhjabsdjfbaj
 the maximum allowed length (20).
        at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modul
e.js:1365:13)
        at SchemaString.SchemaType.doValidate (/Users/shradhakhapra/WebDevelopmen
les/mongoose/lib/schematype.js:1349:7)
        at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modules/mongoose
        at processTicksAndRejections (node:internal/process/task_queues:78:11) {
      properties: [Object],
```

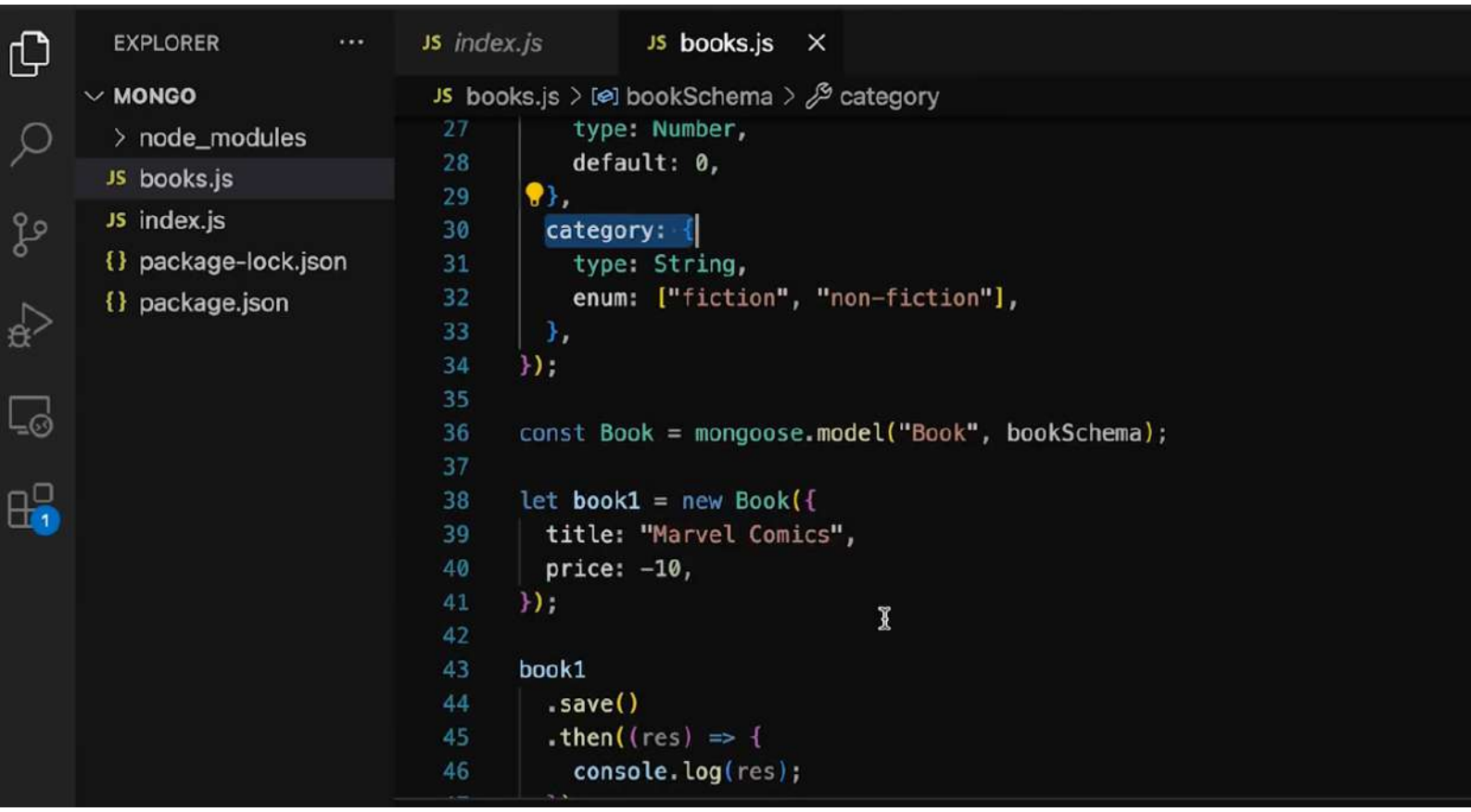


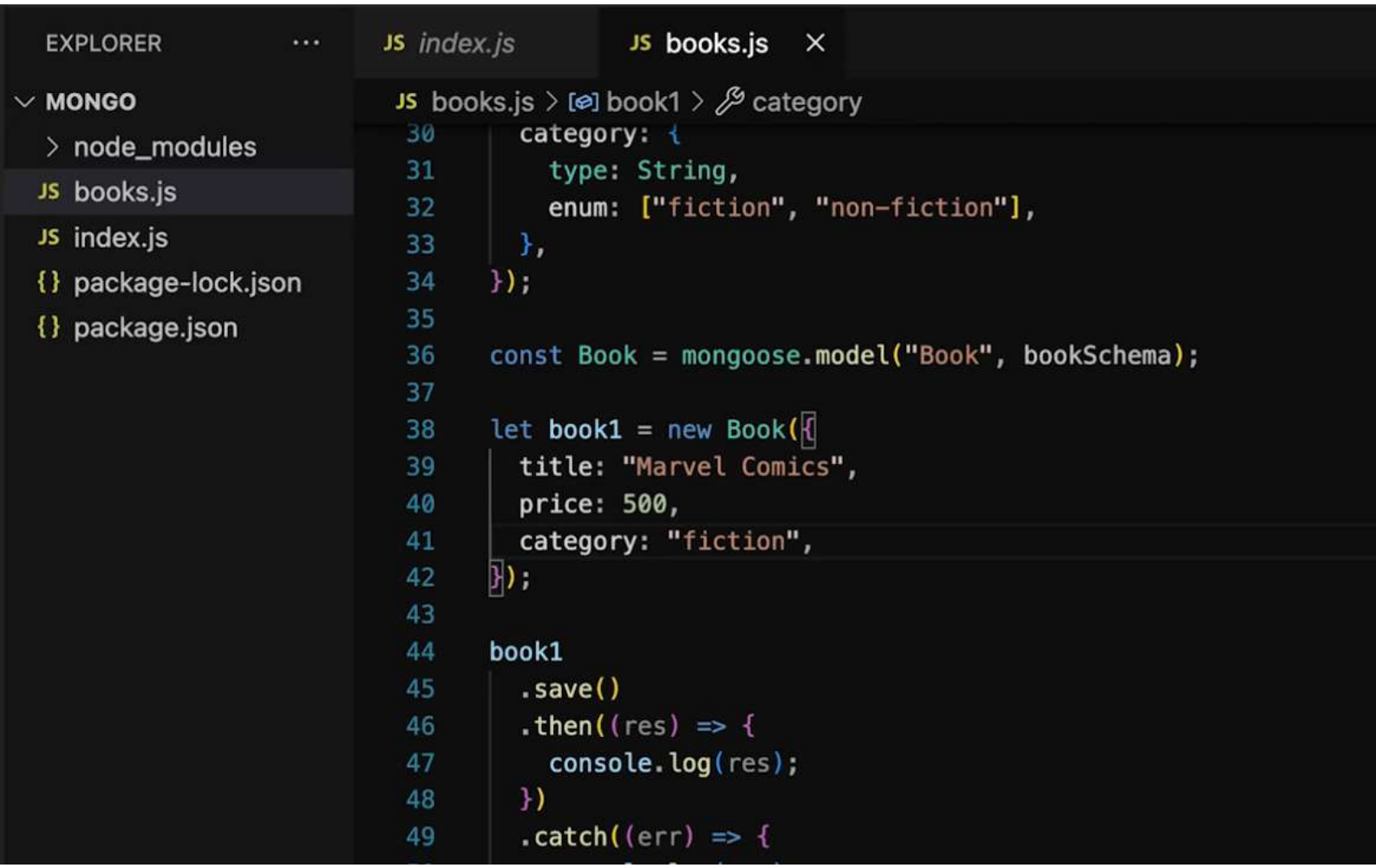


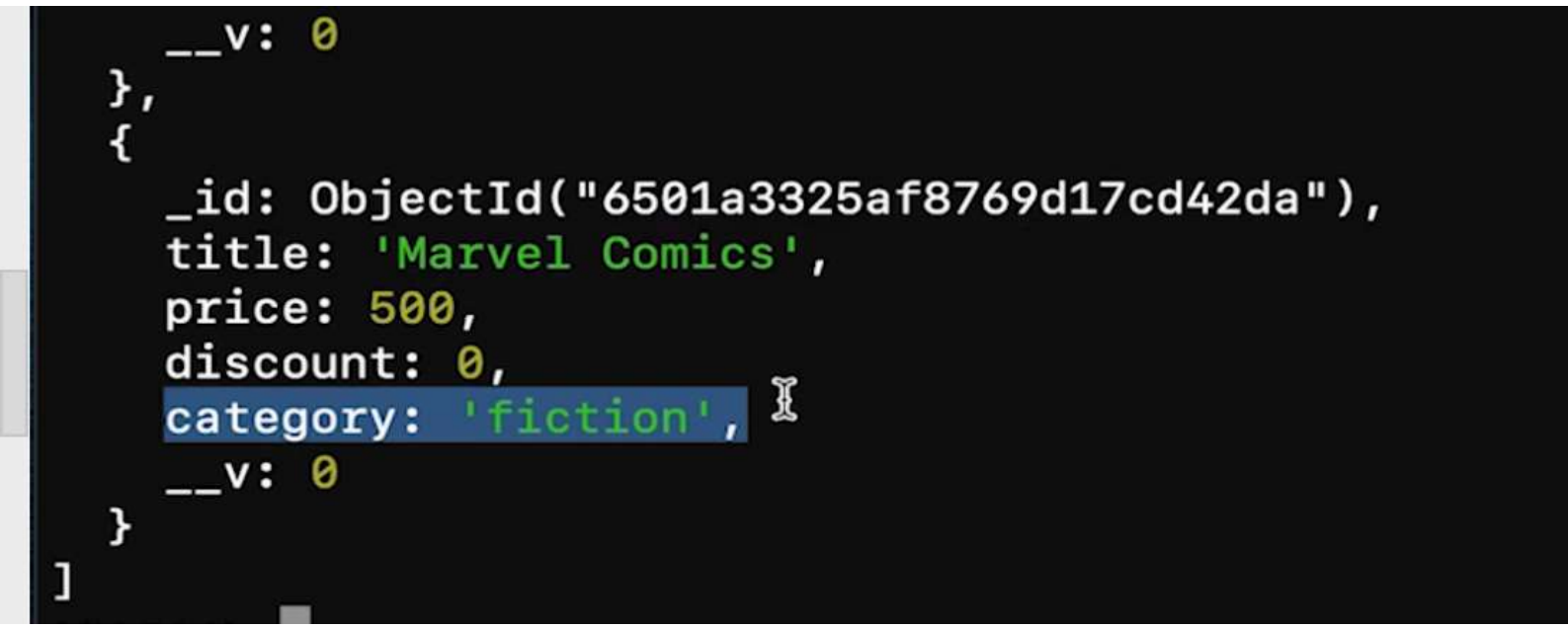
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE zsh

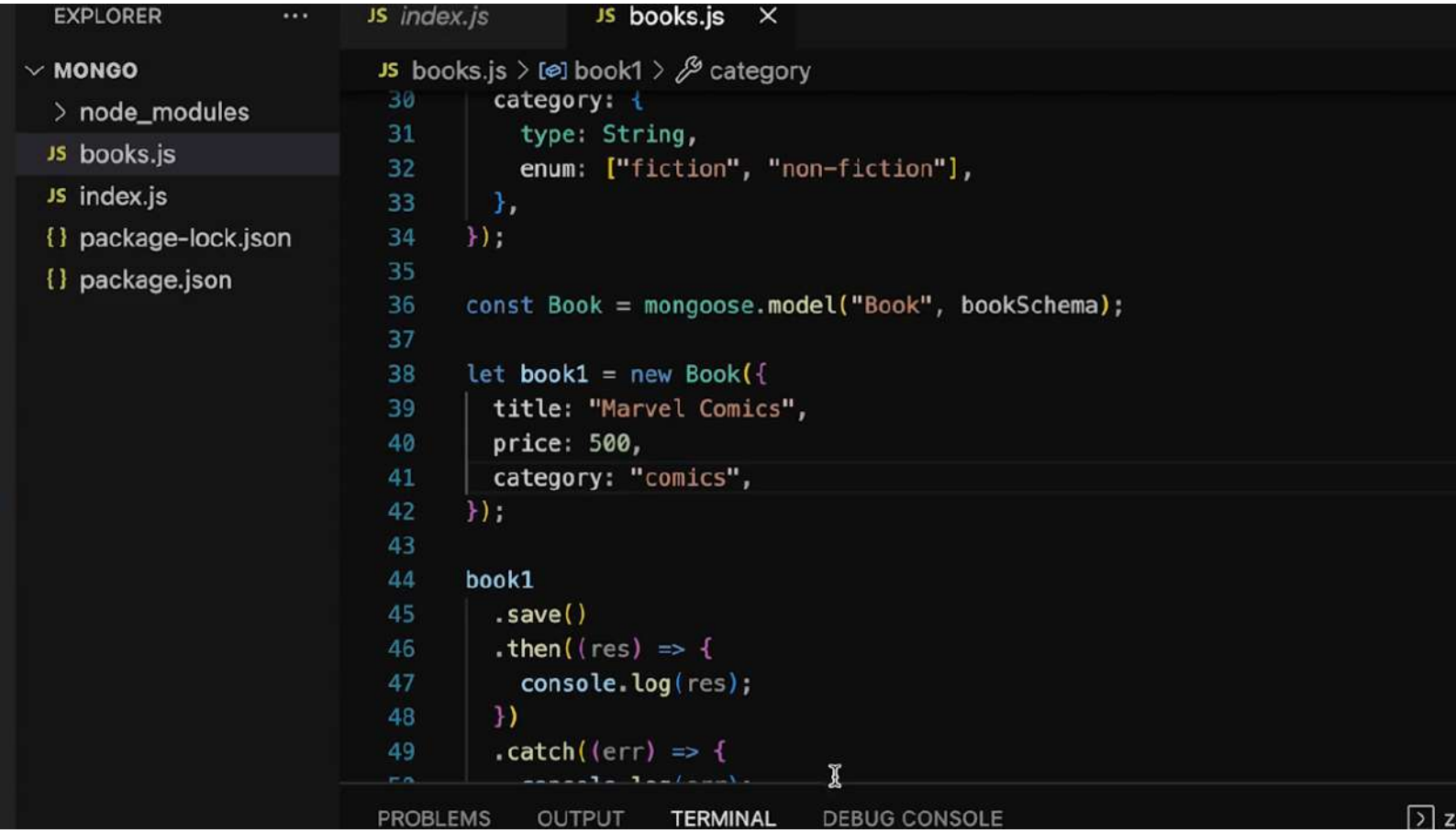
```
      kind: 'maxlength',
      path: 'title',
      value: 'Gone Girl aaaaaaaanbschjasdbfhjabsdjfbajsbdfjba',
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
Error: Book validation failed: price: Path `price` (-10) is less than minimum allowed
    at ValidationError.inspect (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/no
b/error/validation.js:50:26)
    at formatValue (node:internal/util/inspect:763:19)
    at inspect (node:internal/util/inspect:340:10)
    at formatWithOptionsInternal (node:internal/util/inspect:2006:40)
    at formatWithOptions (node:internal/util/inspect:1888:10)
    at console.value (node:internal/console/constructor:323:14)
    at console.log (node:internal/console/constructor:359:61)
    at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:45:13
    at processTicksAndRejections (node:internal/process/task_queues:96:5) {
  errors: {
    price: ValidatorError: Path `price` (-10) is less than minimum allowed value (1
```


enum : ["red", "yellow", "green"]
blue

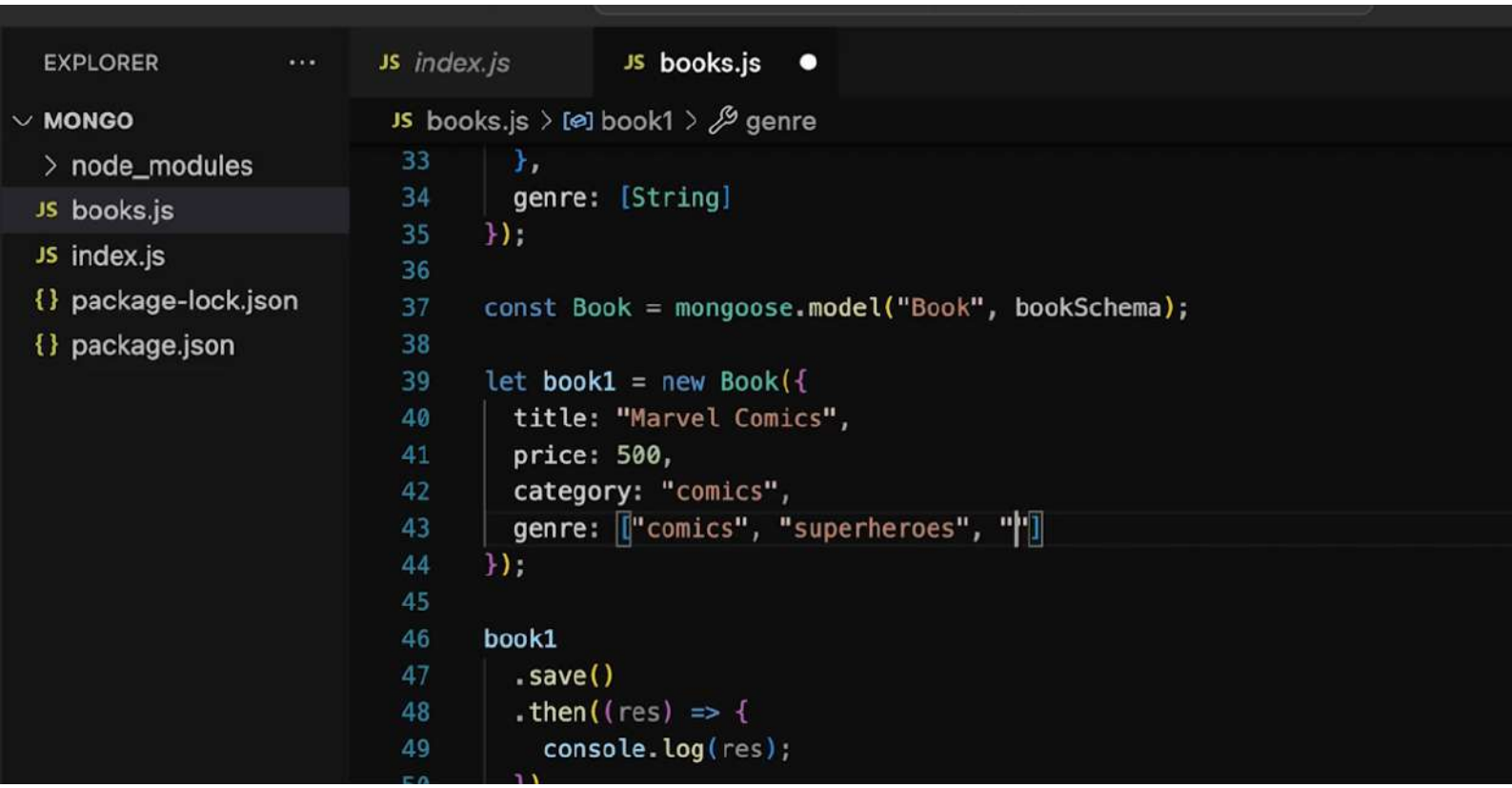

EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
index.js
books.js
books.js > bookSchema > category
type: Number,
default: 0,
},
category: {
type: String,
enum: ["fiction", "non-fiction"],
},
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "Marvel Comics",
price: -10,
});
book1
.save()
.then((res) => {
console.log(res);

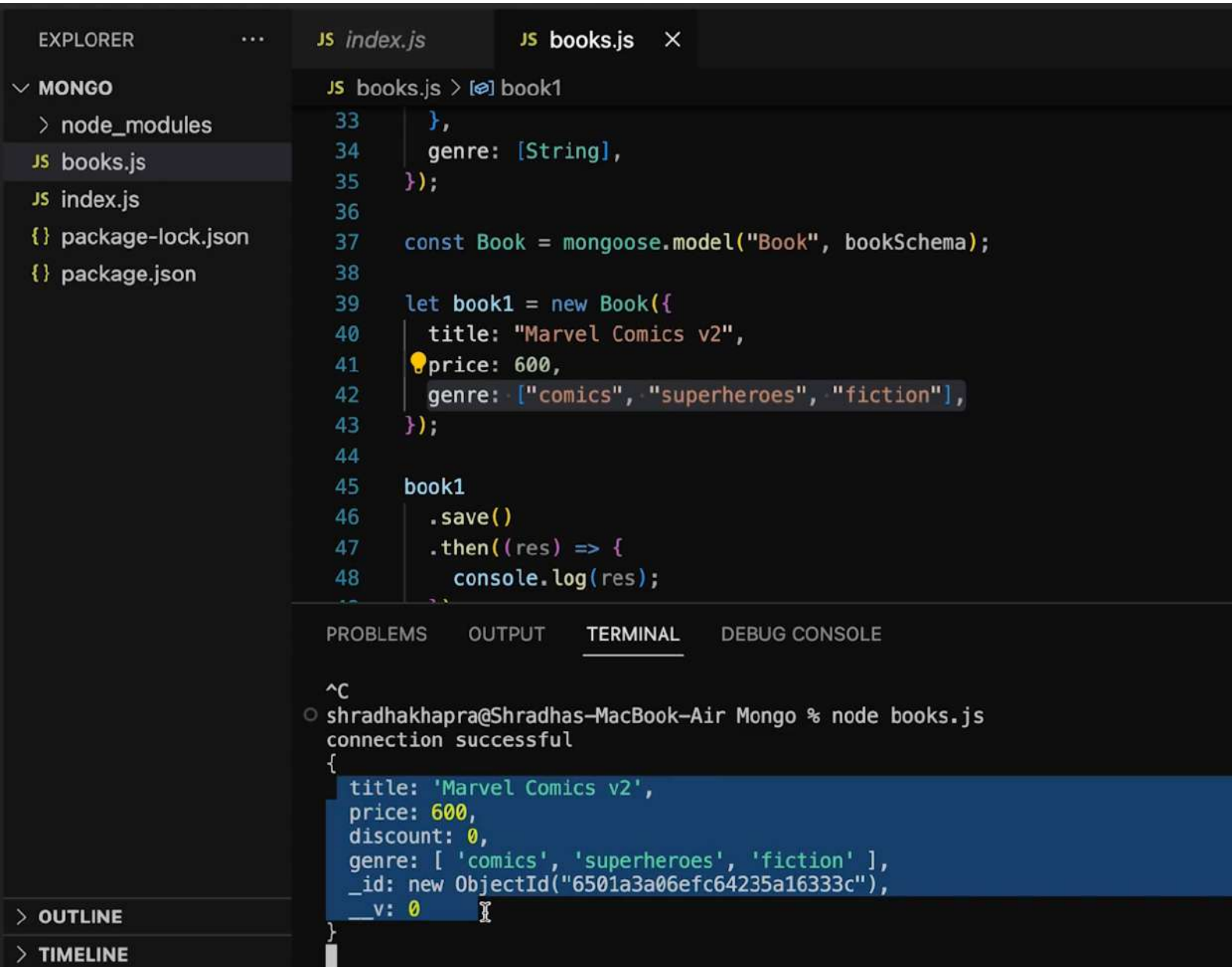

EXPLORER
index.js
books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > book1 > category
category: {
type: String,
enum: ["fiction", "non-fiction"],
},
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "Marvel Comics",
price: 500,
category: "fiction",
});
book1
.save()
.then((res) => {
console.log(res);
})
.catch((err) => {


__v: 0
},
{
_id: ObjectId("6501a3325af8769d17cd42da"),
title: 'Marvel Comics',
price: 500,
discount: 0,
category: 'fiction',
__v: 0
}
]


EXPLORER
index.js
books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > book1 > category
category: {
type: String,
enum: ["fiction", "non-fiction"],
},
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "Marvel Comics",
price: 500,
category: "comics",
});
book1
.save()
.then((res) => {
console.log(res);
})
.catch((err) => {
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE

```
    at console.log (node:internal/console/constructor:359:61)
    at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:50:13
    at processTicksAndRejections (node:internal/process/task_queues:96:5) {
  errors: {
    category: ValidatorError: `comics` is not a valid enum value for path `category`.
        at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modules/mon
e.js:1365:13)
        at SchemaString.SchemaType.doValidate (/Users/shradhakhapra/WebDevelopment/Back
les/mongoose/lib/schematype.js:1349:7)
        at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modules/mongoose/lib/
        at processTicksAndRejections (node:internal/process/task_queues:78:11) {
      properties: [Object],
      kind: 'enum',
      path: 'category',
      value: 'comics',
      reason: undefined,
      [Symbol(mongoose:validatorError)]: true
    }
  },
  _message: 'Book validation failed'
}
connection successful
```


EXPLORER
index.js
books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > book1 > genre
},
genre: [String]
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "Marvel Comics",
price: 500,
category: "comics",
genre: ["comics", "superheroes", ""]
});
book1
.save()
.then((res) => {
console.log(res);


EXPLORER
index.js
books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > book1
},
genre: [String],
});
const Book = mongoose.model("Book", bookSchema);
let book1 = new Book({
title: "Marvel Comics v2",
price: 600,
genre: ["comics", "superheroes", "fiction"],
});
book1
.save()
.then((res) => {
console.log(res);
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
^C
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
connection successful
{
title: 'Marvel Comics v2',
price: 600,
discount: 0,
genre: ['comics', 'superheroes', 'fiction'],
_id: new ObjectId("6501a3a06efc64235a16333c"),
__v: 0
}
OUTLINE
TIMELINE

```
{
  _id: ObjectId("6501a3a06efc64235a16333c"),
  title: 'Marvel Comics v2',
  price: 600,
  discount: 0,
  genre: [ 'comics', 'superheroes', 'fiction' ],
  __v: 0
}
```

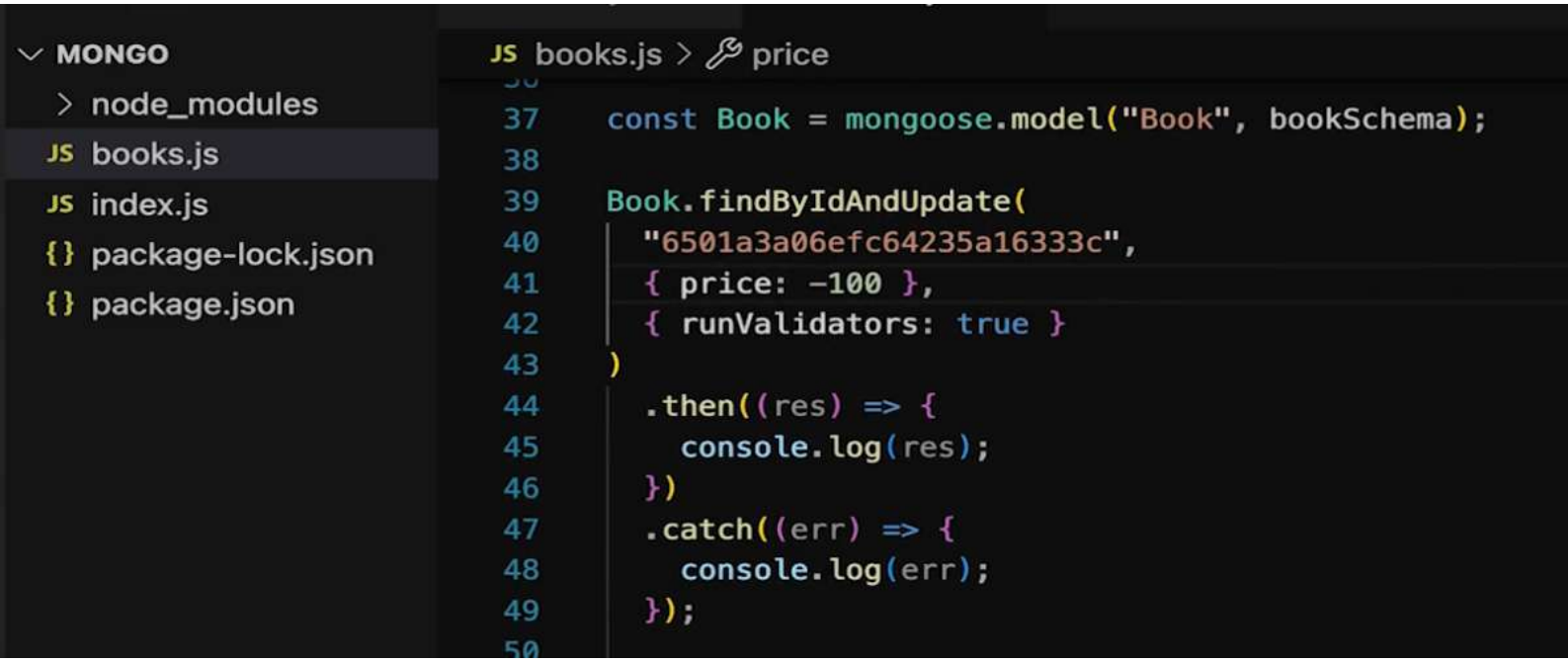
Schema Validations

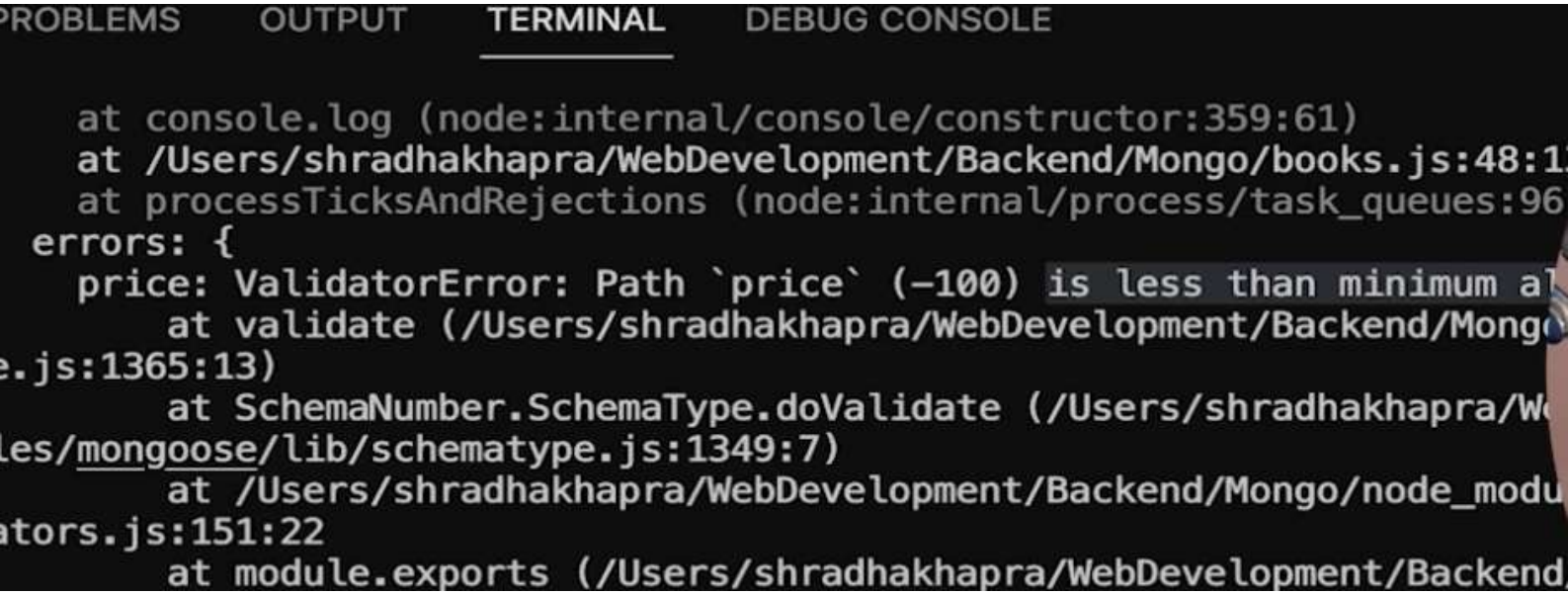
with UPDATE

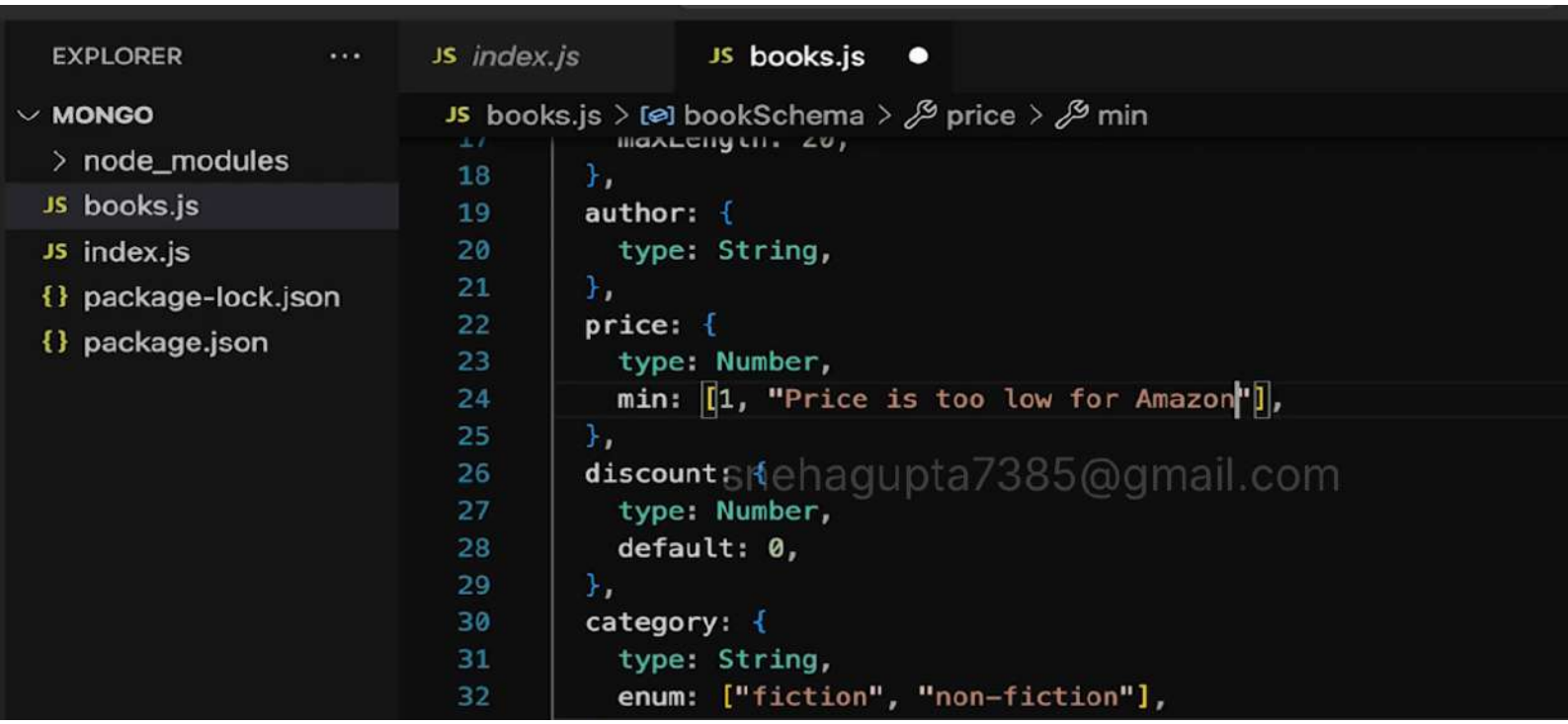
```
35  });
36
37  const Book = mongoose.model("Book", bookSchema);
38  💡
39  Book.findByIdAndUpdate("6501a3a06efc64235a16333c", { price: -500 })
40    .then((res) => {
41      console.log(res);
42    })
43    .catch((err) => {
44      console.log(err);
45    });
46
```

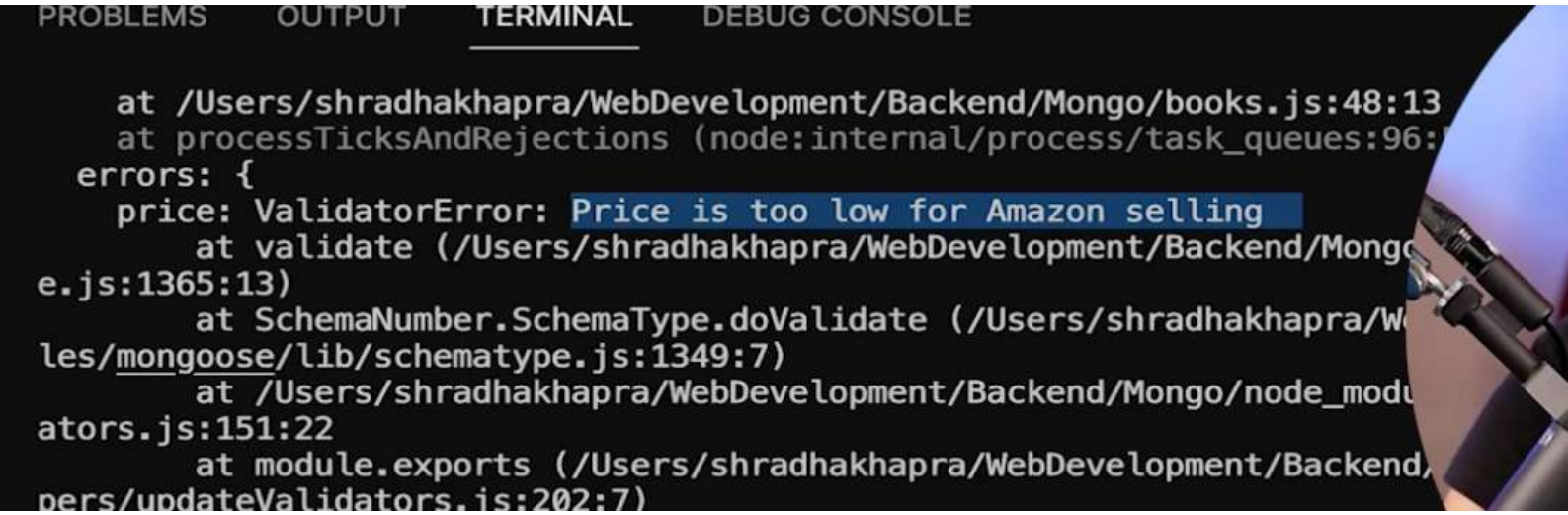
```
{
  _id: ObjectId("6501a3a06efc64235a16333c"),
  title: 'Marvel Comics v2',
  price: -500,
  discount: 0,
  genre: [ 'comics', 'superheroes', 'fiction' ],
  __v: 0
}
```

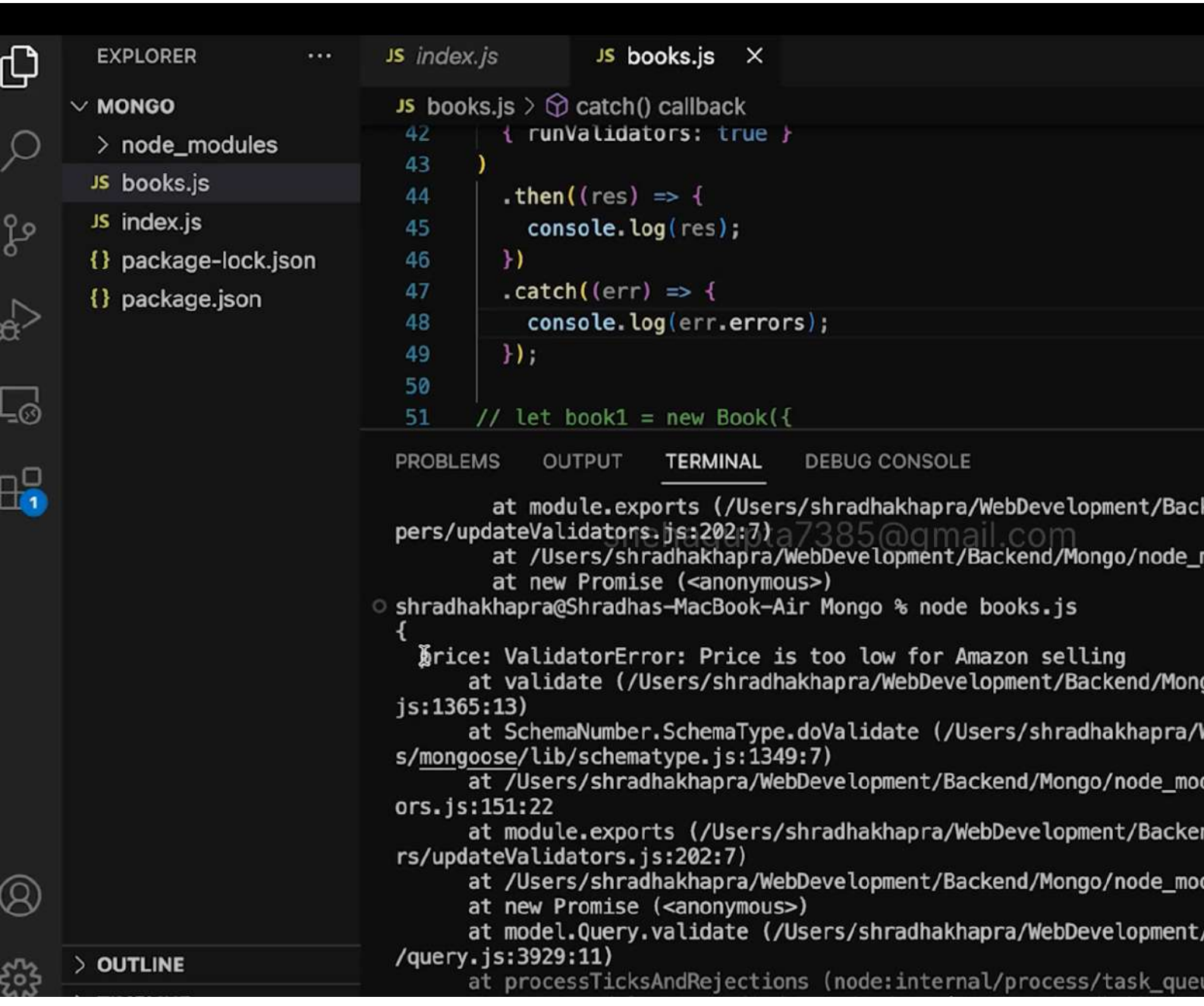
- `[options.returnDocument='before']` «String» Has two possible values, `'before'` and `'after'`. default, it will return the document before the update was applied.
- `[options.lean]` «Object» if truthy, mongoose will return the document as a plain JavaScript objec rather than a mongoose document. See `Query.lean()` and the Mongoose lean tutorial.
- `[options.session=null]` «ClientSession» The session associated with this query. See transaction docs.
- `[options.strict]` «Boolean|String» overwrites the schema's strict mode option
- `[options.timestamps=null]` «Boolean» If set to `false` and schema-level timestamps are enable skip timestamps for this update. Note that this allows you to overwrite timestamps. Does nothing if schema-level timestamps are not set.
- `[options.overwrite=false]` «Boolean» By default, if you don't include any update operators in `update`, Mongoose will wrap `update` in `$set` for you. This prevents you from accidentally overwrit the document. This option tells Mongoose to skip adding `$set`. An alternative to this would be usin Model.findOneAndReplace({ _id: id }, update, options).
- `[options.sort]` «Object|String» if multiple docs are found by the conditions, sets the sort order t choose which doc to update.
- `[options.runValidators]` «Boolean» if true, runs update validators on this command. Upda validators validate the update operation against the model's schema
- `[options.setDefaultsOnInsert=true]` «Boolean» If `setDefaultsOnInsert` and `upsert` mongoose will apply the defaults specified in the model's schema if a new document is cr
- `[options.rawResult]` «Boolean» if true, returns the raw result from the MongoDB driv
- `[options.upsert=false]` «Boolean» if true, and no documents found, insert a new do
- `[options.new=false]` «Boolean» if true, return the modified document rather than th
- `[options.select]` «Object|String» sets the document fields to return.
- `[options.translateAliases=null]` «Boolean» If set to `true`, translates any schema-def

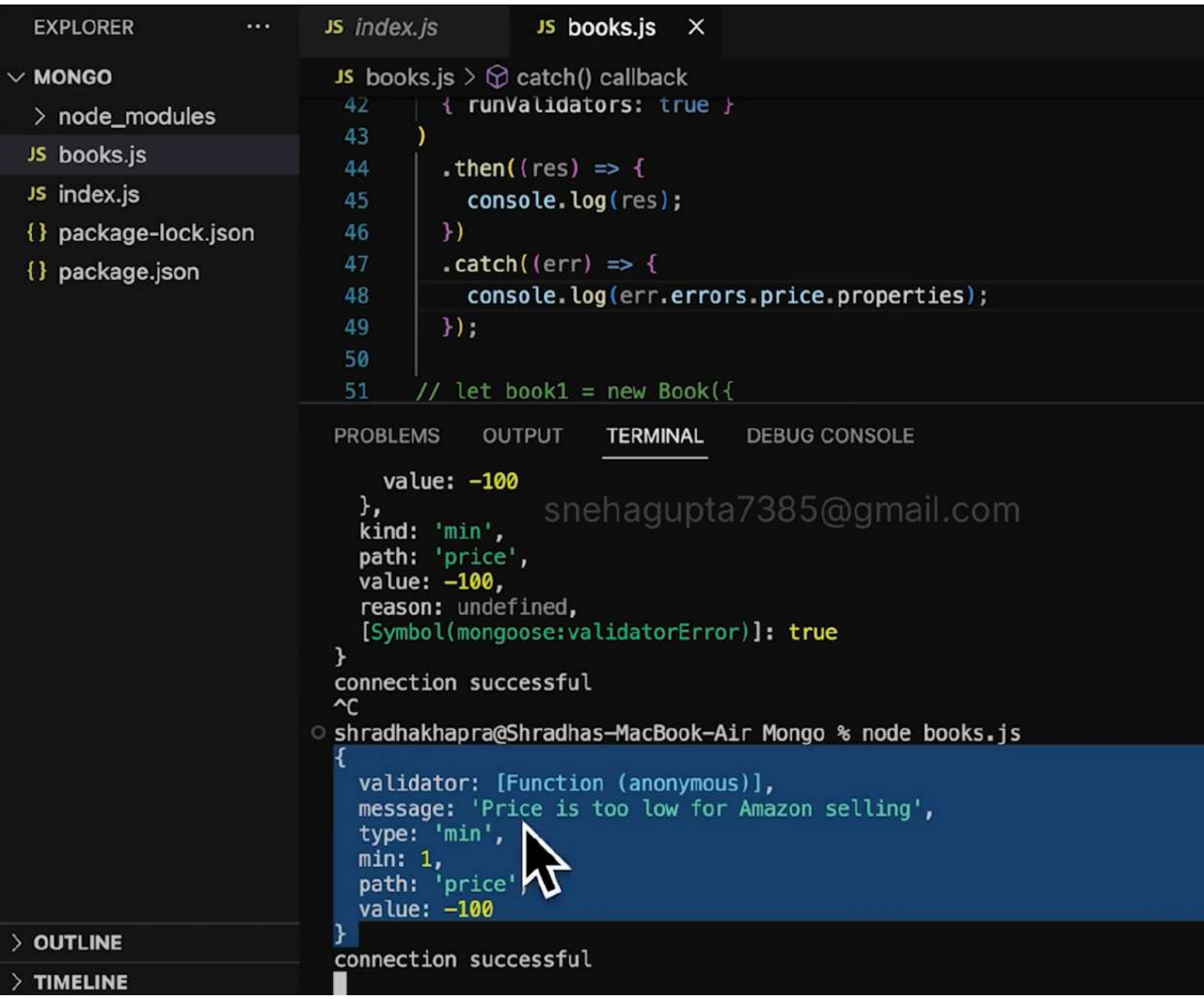

MONGO
node_modules
books.js
index.js
package-lock.json
package.json
books.js > price
const Book = mongoose.model("Book", bookSchema);
Book.findByIdAndUpdate(
"6501a3a06efc64235a16333c",
{ price: -100 },
{ runValidators: true }
)
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err);
});

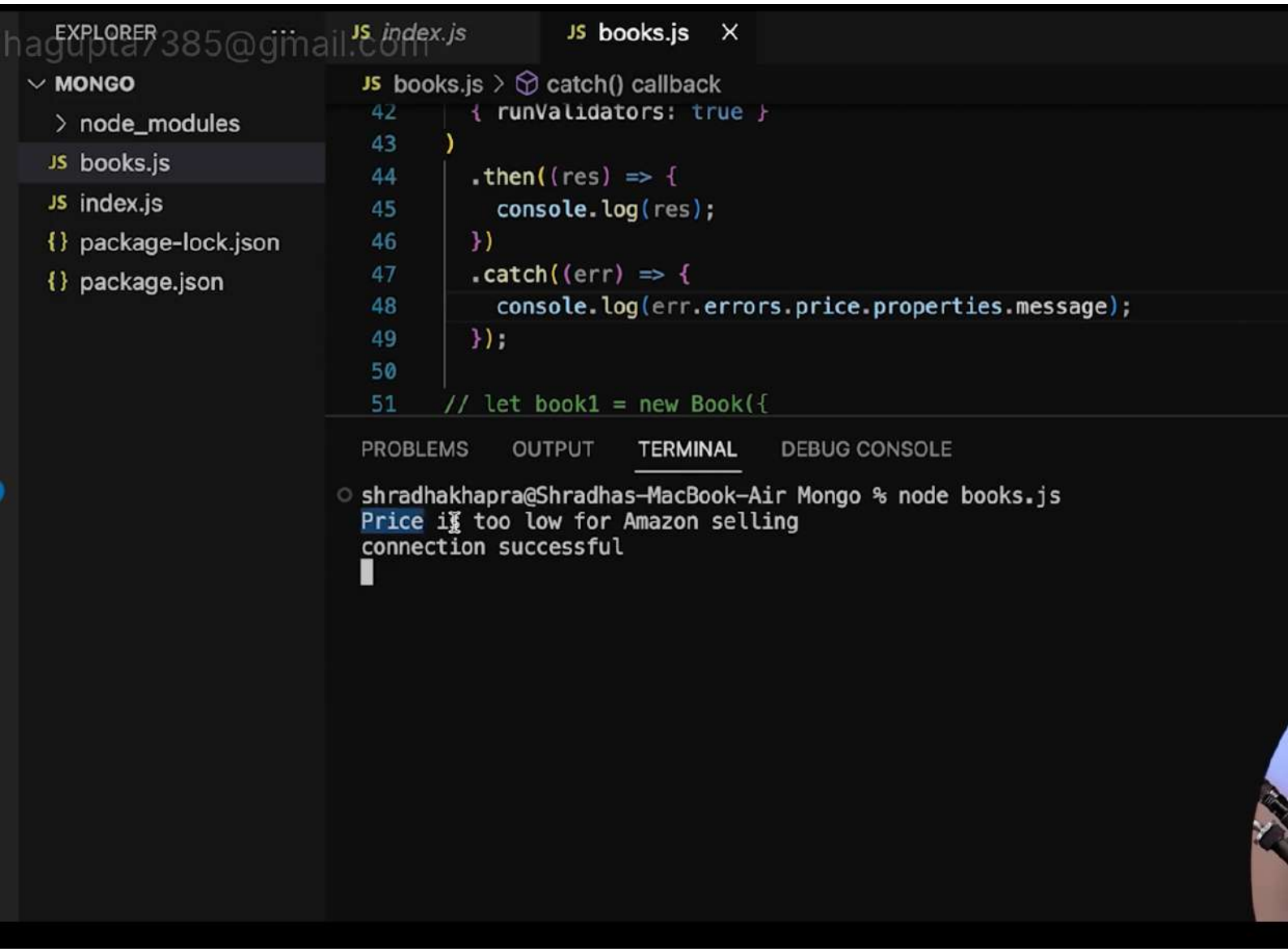

PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
at console.log (node:internal/console/constructor:359:61)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:48:1
at processTicksAndRejections (node:internal/process/task_queues:96
errors: {
price: ValidatorError: Path `price` (-100) is less than minimum al
at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mongo
e.js:1365:13)
at SchemaNumber.SchemaType.doValidate (/Users/shradhakhapra/W
les/mongoose/lib/schematype.js:1349:7)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modu
ators.js:151:22
at module.exports (/Users/shradhakhapra/WebDevelopment/Backend




PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/books.js:48:13
at processTicksAndRejections (node:internal/process/task_queues:96:
errors: {
price: ValidatorError: Price is too low for Amazon selling
at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mongo
e.js:1365:13)
at SchemaNumber.SchemaType.doValidate (/Users/shradhakhapra/W
les/mongoose/lib/schematype.js:1349:7)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_modu
ators.js:151:22
at module.exports (/Users/shradhakhapra/WebDevelopment/Backend/
pers/updateValidators.js:202:7)


EXPLORER
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
OUTLINE
index.js
books.js
books.js > catch() callback
{ runValidators: true }
)
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err.errors);
});
// let book1 = new Book({
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
at module.exports (/Users/shradhakhapra/WebDevelopment/Back
pers/updateValidators.js:202:7)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_
at new Promise (<anonymous>)
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
{
price: ValidatorError: Price is too low for Amazon selling
at validate (/Users/shradhakhapra/WebDevelopment/Backend/Mong
js:1365:13)
at SchemaNumber.SchemaType.doValidate (/Users/shradhakhapra/W
s/mongoose/lib/schematype.js:1349:7)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_mod
ors.js:151:22
at module.exports (/Users/shradhakhapra/WebDevelopment/Backe
rs/updateValidators.js:202:7)
at /Users/shradhakhapra/WebDevelopment/Backend/Mongo/node_mod
at new Promise (<anonymous>)
at model.Query.validate (/Users/shradhakhapra/WebDevelopment/
/query.js:3929:11)
at processTicksAndRejections (node:internal/process/task_que




EXPLORER
JS index.js
JS books.js
MONGO
node_modules
books.js
index.js
package-lock.json
package.json
JS books.js > catch() callback
{ runValidators: true }
)
.then((res) => {
console.log(res);
})
.catch((err) => {
console.log(err.errors.price.properties.message);
});
// let book1 = new Book({
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air Mongo % node books.js
Price is too low for Amazon selling
connection successful

Schema Validations

Handling Errors

```
console.log(err.errors.category.properties.message);
```