LAB 1

❖ CONTENT

- Create database, collection and insert document with MongoDB Compass
- Initialize ExpressJS web app with Handlebars (HBS) view engine
- Install necessary packages with NPM: nodemon, mongoose, body-parser
- Establish database connection in ExpressJS
- Setup relationship between collections (One to Many)
- Implement CRUD features

❖ INTRODUCTION

- MongoDB: NoSQL database, flexible schema, fast query speed, suitable for big data application or content management system
- NoSQL terminology:
  - Collection: table
  - Document: row/record
  - Field: column/property
- ExpressJS: a basic and lightweight backend framework based on NodeJS
- View (template) engine is used to render web pages using template files. Some popular view engines can work with ExpressJS: EJS, HBS, Jade
- CRUD stands for Create, Read, Update, Delete : 4 basic operations of persistent storage (database)
- NPM: package manager for NodeJS packages (modules/libraries)
- MVC stands for Model, View, Controller : a popular design architecture

➢ DEMO PROJECT

- Database name: **web**
- Collections: **categories** & **products**
- Relationship: One (**categories**) to Many (**products**)
  - categories.**_id** : Primary Key
  - products.**category** : Foreign Key
  - *Note:* **_id** is auto generated value, **objectId** is type of Primary Key

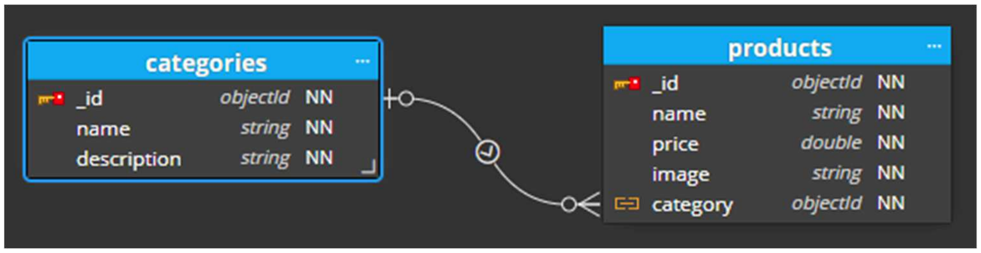
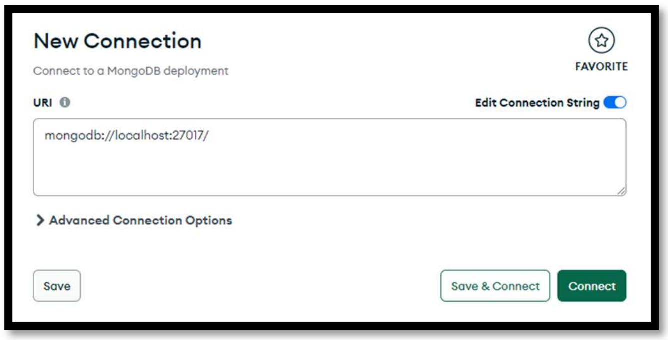


Figure 1 - Database diagram

- Features to implement first: CRUD

❖ INSTRUCTION

1. Open MongoDB Compass and click Connect to make connection to local MongoDB server



New Connection

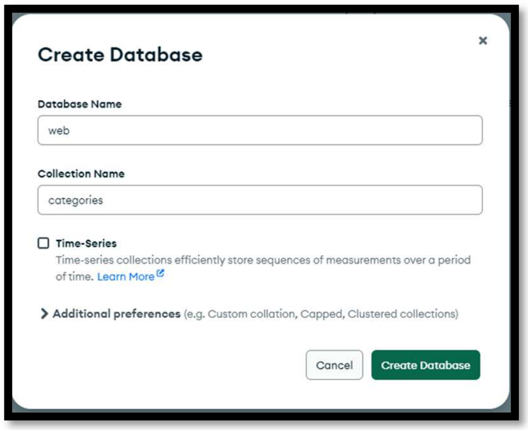
Connect to a MongoDB deployment

URI

mongodb://localhost:27017/

Figure 2 - Connect to local MongoDB server

2. Create new database, new collections and populate (insert) new documents


Create Database

Database Name: web

Collection Name: categories

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

> Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Figure 3 - Create database and collection

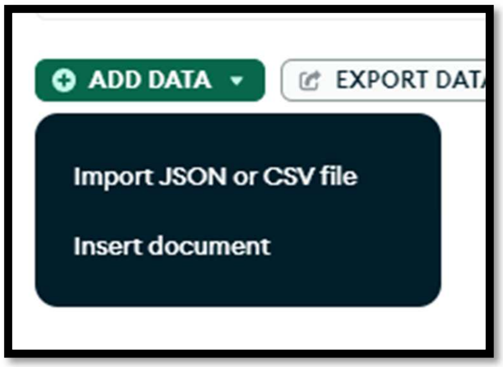
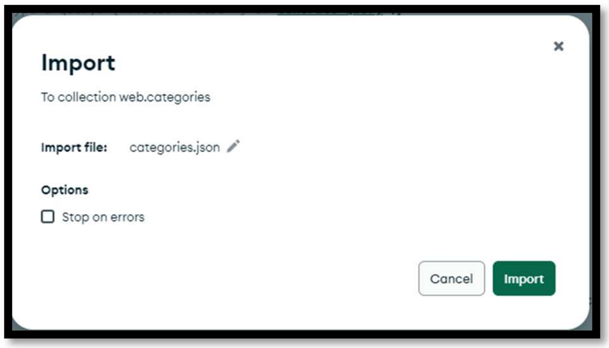


Figure 4 - Add data to collections by importing JSON/CSV or inserting documents


Import

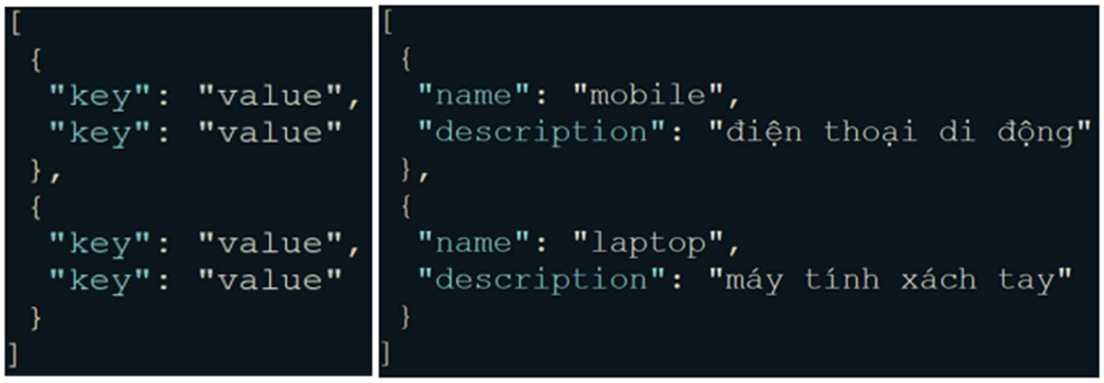
To collection web.categories

Import file: categories.json

Options

☐ Stop on errors

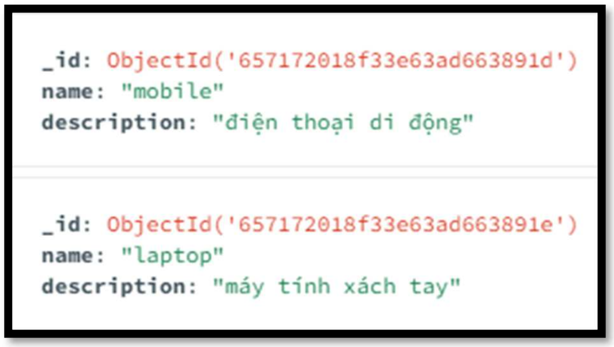
Figure 5 - Import data by JSON file



```
[
  {
    "key": "value",
    "key": "value"
  },
  {
    "key": "value",
    "key": "value"
  }
]
```

```
[
  {
    "name": "mobile",
    "description": "điện thoại di động"
  },
  {
    "name": "laptop",
    "description": "máy tính xách tay"
  }
]
```

Figure 6 - Insert data by JSON format (MongoDB will automatically determine data type based on input value)



```
_id: ObjectId('657172018f33e63ad663891d')
name: "mobile"
description: "điện thoại di động"

_id: ObjectId('657172018f33e63ad663891e')
name: "laptop"
description: "máy tính xách tay"
```

*Figure 7 – **Categories** collection*

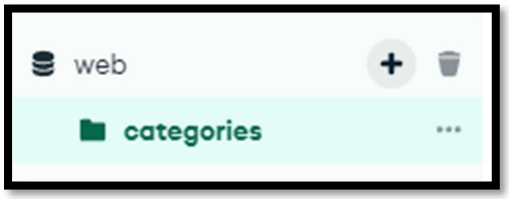
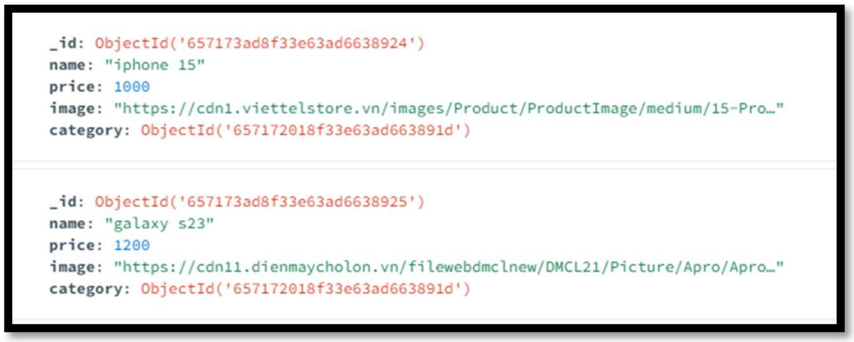


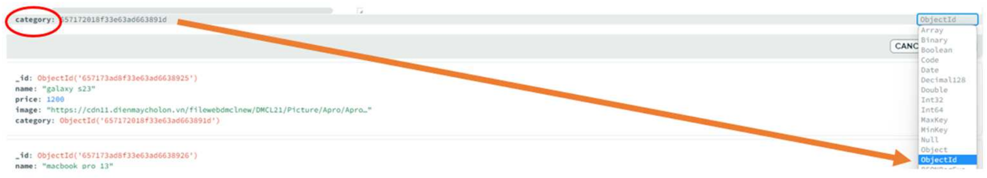
*Figure 8 - Create new collection **Products** (click on **+** icon)*



```
_id: ObjectId('657173ad8f33e63ad6638924')
name: "iphone 15"
price: 1000
image: "https://cdn1.viettelstore.vn/images/Product/ProductImage/medium/15-Pro…"
category: ObjectId('657172018f33e63ad663891d')

_id: ObjectId('657173ad8f33e63ad6638925')
name: "galaxy s23"
price: 1200
image: "https://cdn11.dienmaycholon.vn/filewebdmclnew/DMCL21/Picture/Apro/Apro…"
category: ObjectId('657172018f33e63ad663891d')
```

*Figure 9 - **Products** collection*



```
category: 657172018f33e63ad663891d

_id: ObjectId('657173ad8f33e63ad6638925')
name: "galaxy s23"
price: 1200
image: "https://cdn11.dienmaycholon.vn/filewebdmclnew/DMCL21/Picture/Apro/Apro…"
category: ObjectId('657172018f33e63ad663891d')

_id: ObjectId('657173ad8f33e63ad6638926')
name: "macbook pro 13"
```

*Figure 10 - Change type of a field (**category** value in **products** must match with **_id** value in **categories**)*

3. Open VS Code and select 1 folder to saving project source code

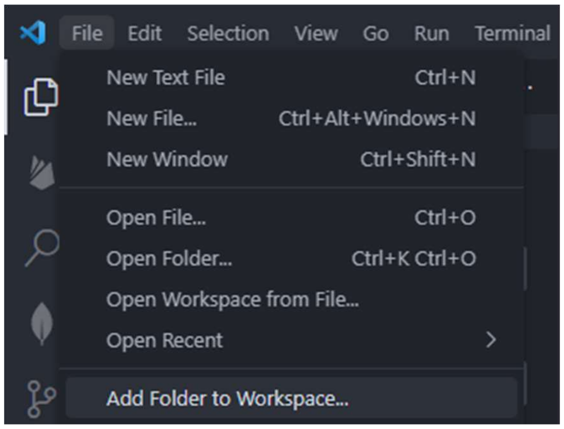

File
Edit
Selection
View
Go
Run
Terminal
New Text File Ctrl+N
New File... Ctrl+Alt+Windows+N
New Window Ctrl+Shift+N
Open File... Ctrl+O
Open Folder... Ctrl+K Ctrl+O
Open Workspace from File...
Open Recent
Add Folder to Workspace...

Figure 11 - Add 1 folder to workspace for saving codes

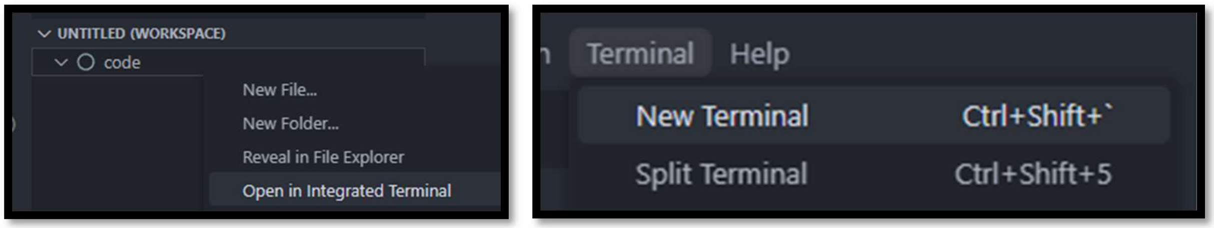


Figure 12 – Open Terminal by 1 of 2 above methods

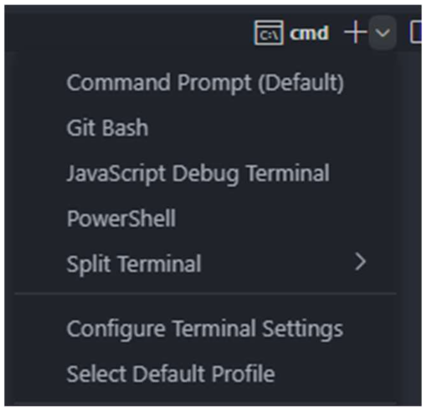


Figure 13 - Select Command Prompt and Set it as Default Profile (only 1st time)

4. Initialize new ExpressJS project by typing commands in Terminal

```
npx express-generator --view=hbs
```

Figure 14 - Create new ExpressJS project with HBS view engine

```
npm install
npm install nodemon -g
npm install mongoose --save
npm install body-parser --save
```

Figure 15 - Install necessary packages for project

```
npm install nodemon mongoose body-parser
```

Figure 16 - Install necessary packages (shorthand)

```
nodemon : auto reload server after code update
mongoose : connect and manage MongoDB database
body-parser : retrieve client-side input data
```

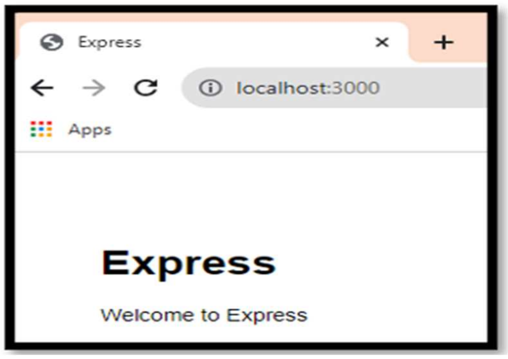
Figure 17 - Purspose of each package

```
echo node_modules > .gitignore
```

*Figure 18 - Create file **.gitignore** (shorthand) to ignore **node_modules** folder when pushing code to GitHub*

```
nodemon
```

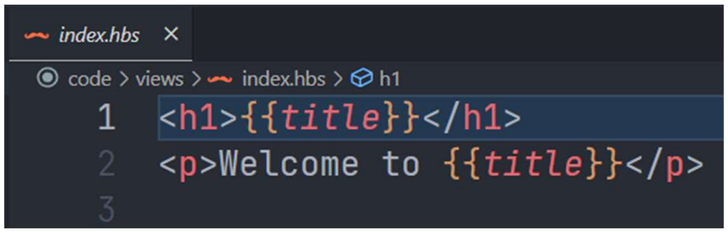
*Figure 19 - Run web project with auto-reload when coding (use **npm start** after finish)*



Express

Welcome to Express

*Figure 20 - Open browser and type "**localhost:3000**" to see web homepage (Note: 3000 is default port)*



```
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
```

Figure 21 - Modify this code to see the change effect

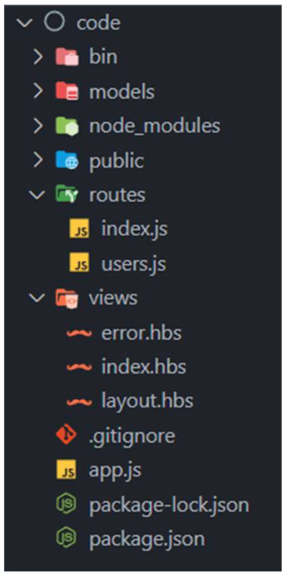
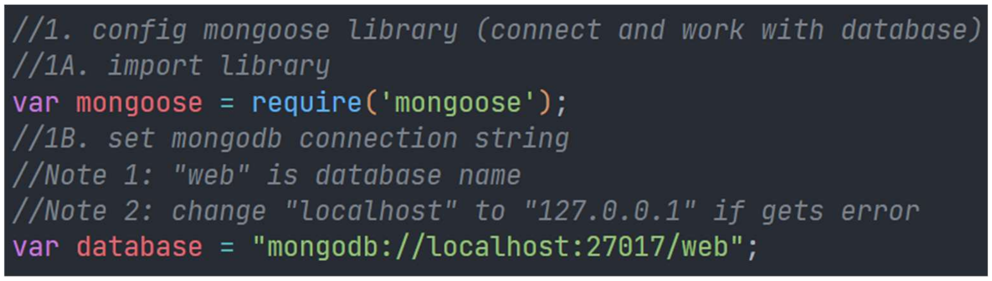


Figure 22 - ExpressJS project structure

- **models** folder (*create manually*) : Model
- **views** folder: View
- **routes** folder: Controller
- **app.js**: main file to run web app
- **package.json**: declaration of all package names be used in project
- **node_modules** folder: store all package folders
- **public** folder: stores static files (such as images, documents, css, js, …)

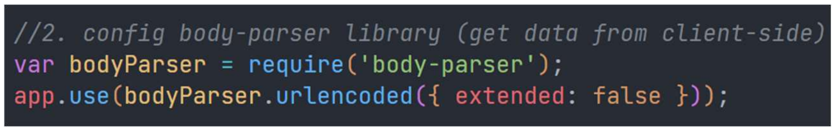
➢ **Coding order:**
app.js ➔ *models* ➔ *routes* ➔ *views*

5. Import and config packages, create routers in file **app.js**



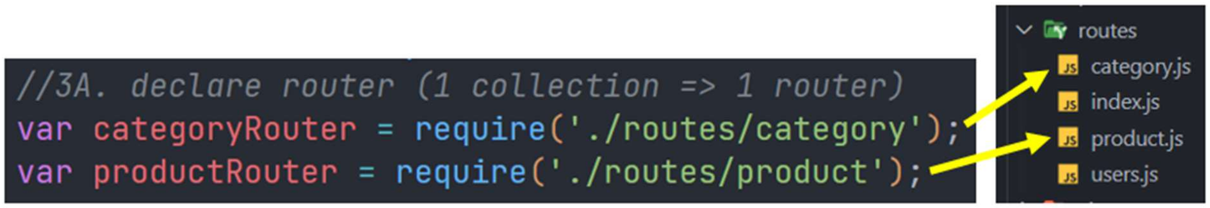
```
//1. config mongoose library (connect and work with database)
//1A. import library
var mongoose = require('mongoose');
//1B. set mongodb connection string
//Note 1: "web" is database name
//Note 2: change "localhost" to "127.0.0.1" if gets error
var database = "mongodb://localhost:27017/web";
```

Figure 23 – Import & config mongoose **package** (remember to declare database name)



```
//2. config body-parser library (get data from client-side)
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
```

Figure 24 - Import & config **body-parser** package


```
//3A. declare router (1 collection => 1 router)
var categoryRouter = require('./routes/category');
var productRouter = require('./routes/product');
```

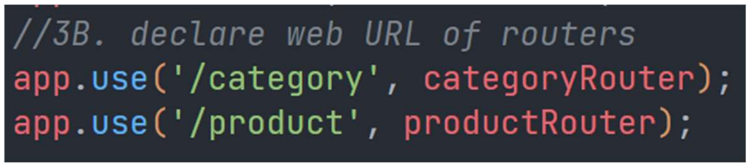

```
//3B. declare web URL of routers
app.use('/category', categoryRouter);
app.use('/product', productRouter);
```

Figure 25 - Declare relative web url of routers

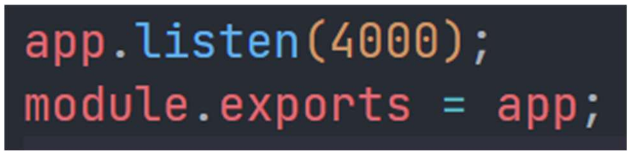

```
app.listen(4000);
module.exports = app;
```

Figure 26 - Modify web server port (if necessary, such as cloud deployment)

6. Create and code **models (M)**

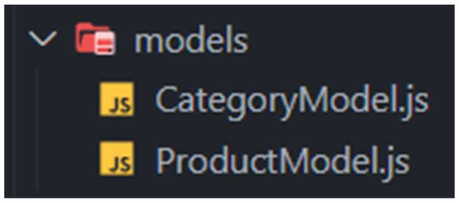


Figure 27 - Create models (1 model for 1 collection)

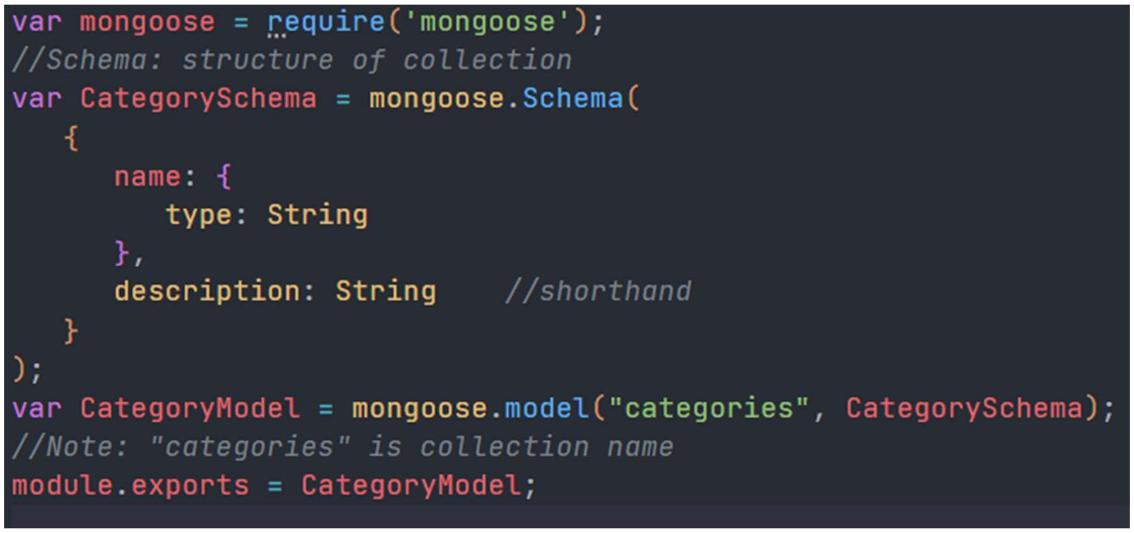
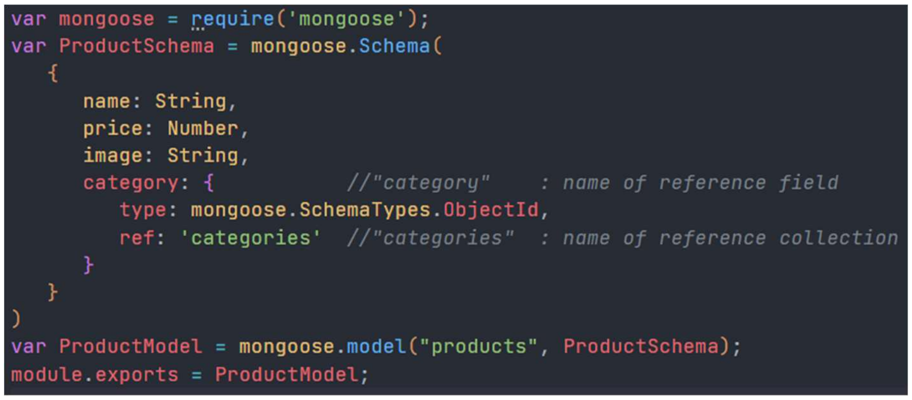

```
var mongoose = require('mongoose');
//Schema: structure of collection
var CategorySchema = mongoose.Schema(
   {
      name: {
         type: String
      },
      description: String    //shorthand
   }
);
var CategoryModel = mongoose.model("categories", CategorySchema);
//Note: "categories" is collection name
module.exports = CategoryModel;
```

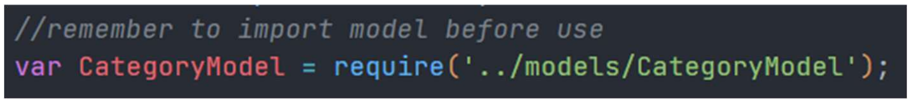
*Figure 28 - Declare **Category** model (**CategoryModel.js**)*



```
var mongoose = require('mongoose');
var ProductSchema = mongoose.Schema(
   {
      name: String,
      price: Number,
      image: String,
      category: {           //"category"    : name of reference field
         type: mongoose.SchemaTypes.ObjectId,
         ref: 'categories'  //"categories"  : name of reference collection
      }
   }
)
var ProductModel = mongoose.model("products", ProductSchema);
module.exports = ProductModel;
```

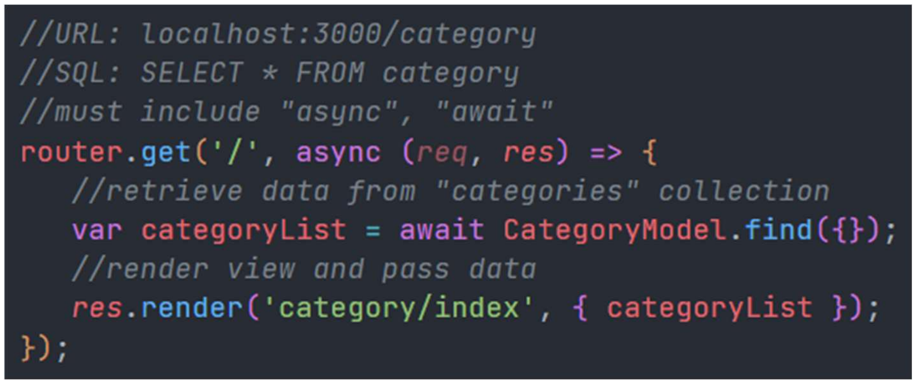
*Figure 29 - Declare **Product** model (**ProductModel.js**)*

7. Create and code **routes (C)**



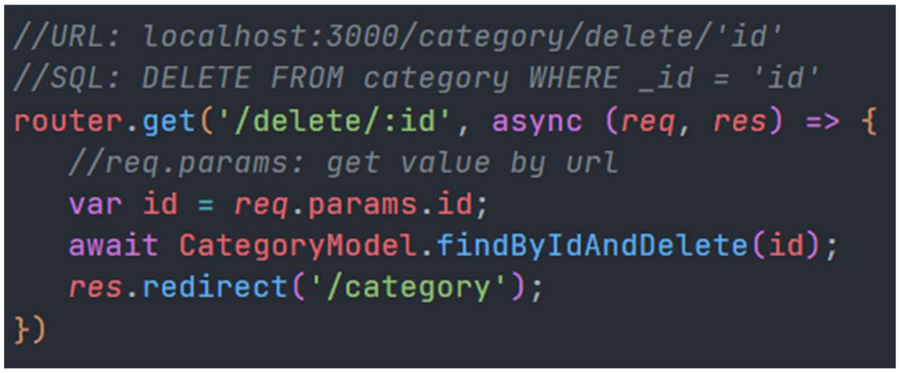
```
//remember to import model before use
var CategoryModel = require('../models/CategoryModel');
```

*Figure 30 - Import model before use (**category.js**)*



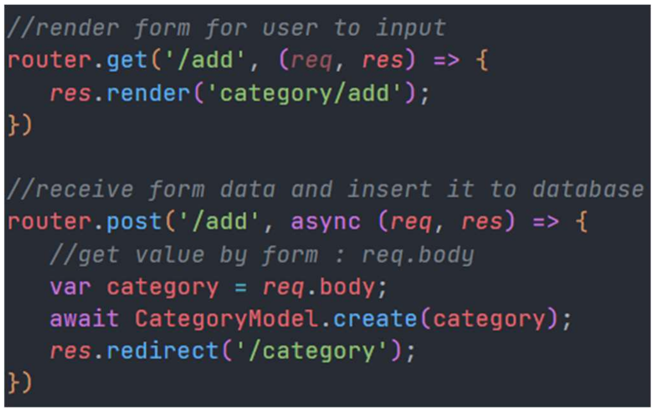
```
//URL: localhost:3000/category
//SQL: SELECT * FROM category
//must include "async", "await"
router.get('/', async (req, res) => {
   //retrieve data from "categories" collection
   var categoryList = await CategoryModel.find({});
   //render view and pass data
   res.render('category/index', { categoryList });
});
```

*Figure 31 - **READ** feature (**category.js**)*



```
//URL: localhost:3000/category/delete/'id'
//SQL: DELETE FROM category WHERE _id = 'id'
router.get('/delete/:id', async (req, res) => {
   //req.params: get value by url
   var id = req.params.id;
   await CategoryModel.findByIdAndDelete(id);
   res.redirect('/category');
})
```

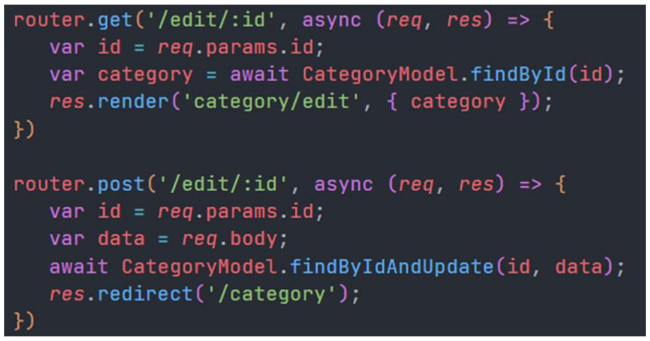
*Figure 32 - **DELETE** feature **(category.js)***



```
//render form for user to input
router.get('/add', (req, res) => {
   res.render('category/add');
})

//receive form data and insert it to database
router.post('/add', async (req, res) => {
   //get value by form : req.body
   var category = req.body;
   await CategoryModel.create(category);
   res.redirect('/category');
})
```

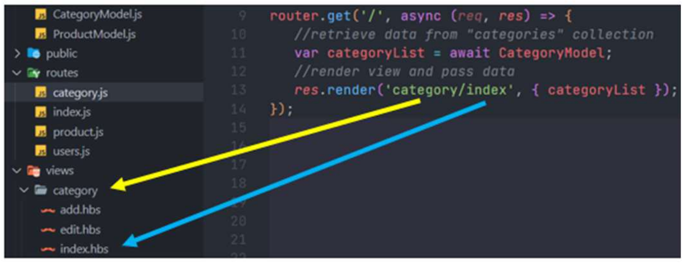
*Figure 33 - **CREATE** feature (**category.js**)*



```
router.get('/edit/:id', async (req, res) => {
   var id = req.params.id;
   var category = await CategoryModel.findById(id);
   res.render('category/edit', { category });
})

router.post('/edit/:id', async (req, res) => {
   var id = req.params.id;
   var data = req.body;
   await CategoryModel.findByIdAndUpdate(id, data);
   res.redirect('/category');
})
```

*Figure 34 - **UPDATE** feature (**category.js**)*

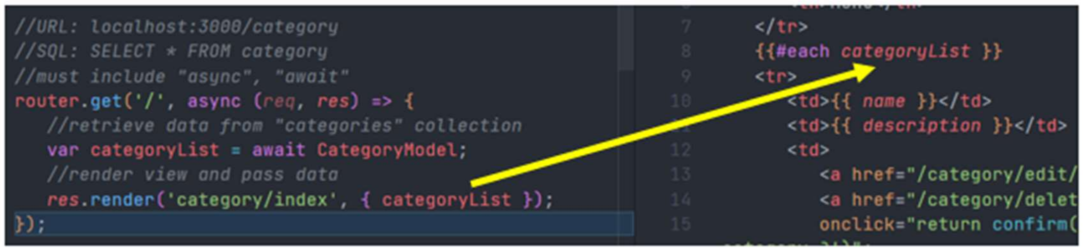


```
CategoryModel.js
ProductModel.js
> public
v routes
    category.js
    index.js
    product.js
    users.js
v views
  v category
      add.hbs
      edit.hbs
      index.hbs
```

```
router.get('/', async (req, res) => {
   //retrieve data from "categories" collection
   var categoryList = await CategoryModel;
   //render view and pass data
   res.render('category/index', { categoryList });
});
```

*Figure 35 – **Controller** (back-end) renders **View** (front-end)*

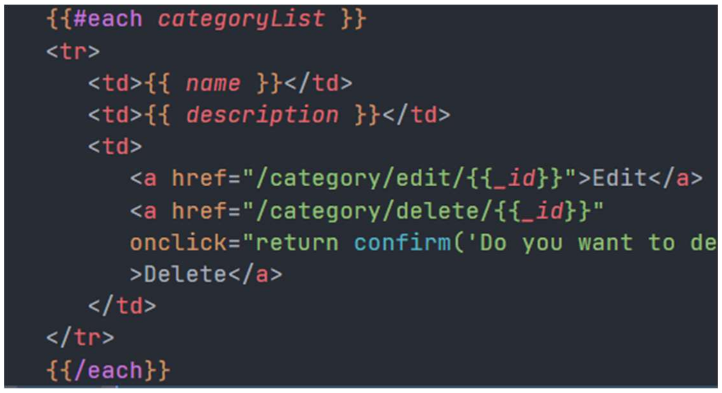
8. Create and code **views** (**V**)



```
//URL: localhost:3000/category
//SQL: SELECT * FROM category
//must include "async", "await"
router.get('/', async (req, res) => {
   //retrieve data from "categories" collection
   var categoryList = await CategoryModel;
   //render view and pass data
   res.render('category/index', { categoryList });
});
```

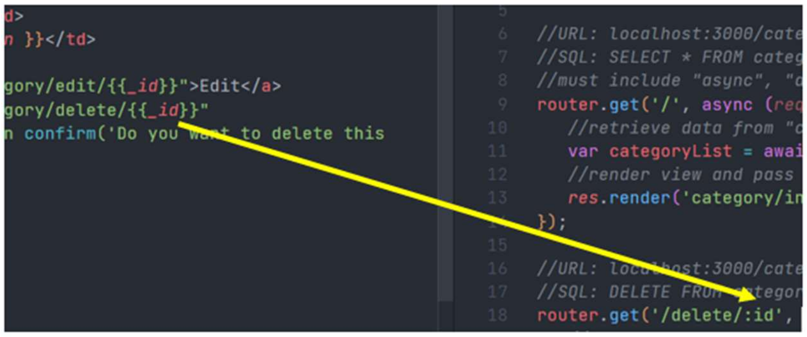
```
</tr>
{{#each categoryList }}
<tr>
   <td>{{ name }}</td>
   <td>{{ description }}</td>
   <td>
      <a href="/category/edit/
      <a href="/category/delet
      onclick="return confirm(
```

*Figure 36 - **Controller** pass data to **View***



```
{{#each categoryList }}
<tr>
   <td>{{ name }}</td>
   <td>{{ description }}</td>
   <td>
      <a href="/category/edit/{{_id}}">Edit</a>
      <a href="/category/delete/{{_id}}"
      onclick="return confirm('Do you want to de
      >Delete</a>
   </td>
</tr>
{{/each}}
```

*Figure 37 - Use foreach loop **(#each)** to load data to table (category index page)*



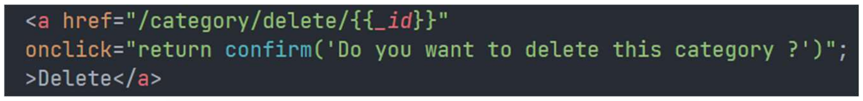
```
d>
n }}</td>

gory/edit/{{_id}}">Edit</a>
gory/delete/{{_id}}"
n confirm('Do you want to delete this
```

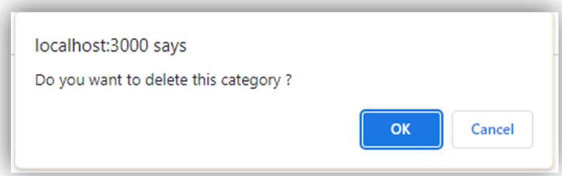
```
//URL: localhost:3000/cate
//SQL: SELECT * FROM categ
//must include "async", "a
router.get('/', async (req
   //retrieve data from "c
   var categoryList = awai
   //render view and pass
   res.render('category/in
});

//URL: localhost:3000/cate
//SQL: DELETE FROM categor
router.get('/delete/:id',
```

*Figure 38 - **View** pass data to **Controller** by **url (req.params)***



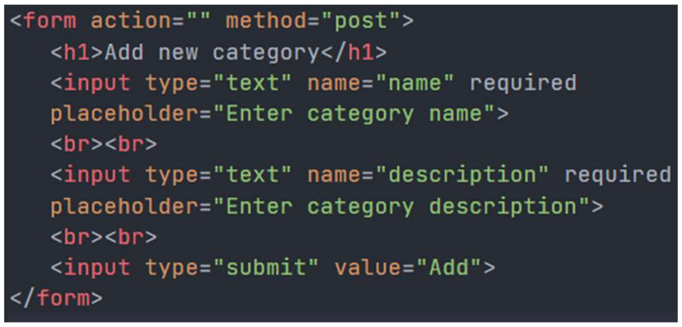
```
<a href="/category/delete/{{_id}}"
onclick="return confirm('Do you want to delete this category ?')";
>Delete</a>
```



localhost:3000 says

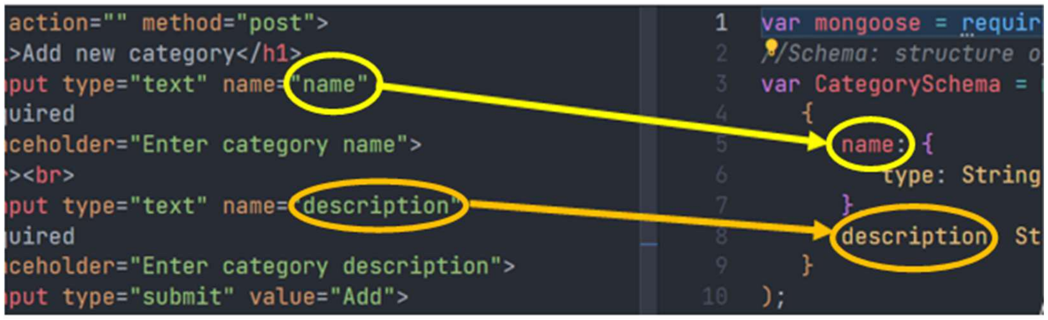
Do you want to delete this category ?

Figure 39 - Display confirm dialog when delete data



```
<form action="" method="post">
   <h1>Add new category</h1>
   <input type="text" name="name" required
   placeholder="Enter category name">
   <br><br>
   <input type="text" name="description" required
   placeholder="Enter category description">
   <br><br>
   <input type="submit" value="Add">
</form>
```

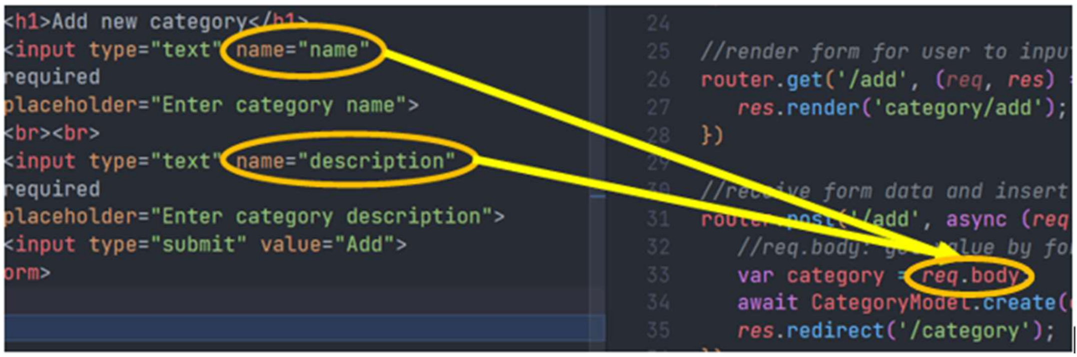
Figure 40 - Add new category form



```
action="" method="post">
>Add new category</h1>
put type="text" name="name"
uired
ceholder="Enter category name">
><br>
put type="text" name="description"
uired
ceholder="Enter category description">
put type="submit" value="Add">
```

```
var mongoose = requir
//Schema: structure o
var CategorySchema =
  {
    name: {
      type: String
    }
    description: St
  }
);
```

Figure 41 - ***name*** *attribute in form* ***View*** *must match with* ***field*** *name (column) in* ***Model***

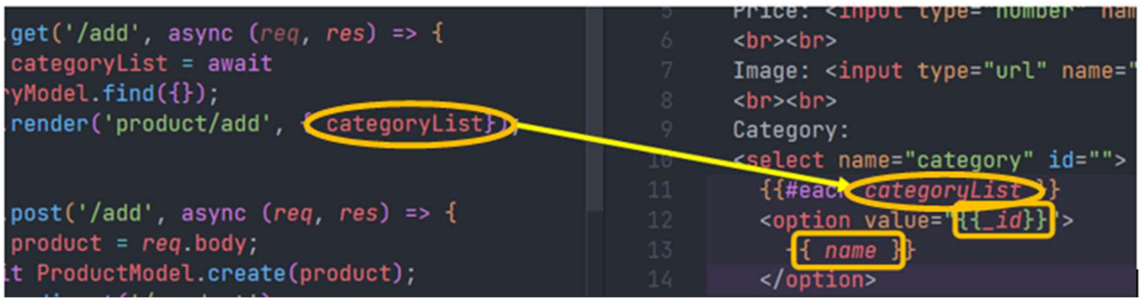


```
<h1>Add new category</h1>
<input type="text" name="name"
required
placeholder="Enter category name">
<br><br>
<input type="text" name="description"
required
placeholder="Enter category description">
<input type="submit" value="Add">
orm>
```

```
//render form for user to inpu
router.get('/add', (req, res)
  res.render('category/add');
})

//receive form data and insert
router.post('/add', async (req
  //req.body: get value by fo
  var category = req.body
  await CategoryModel.create(
  res.redirect('/category');
```

Figure 42 - ***View*** *pass data to* ***Controller*** *using* ***form (req.body)***

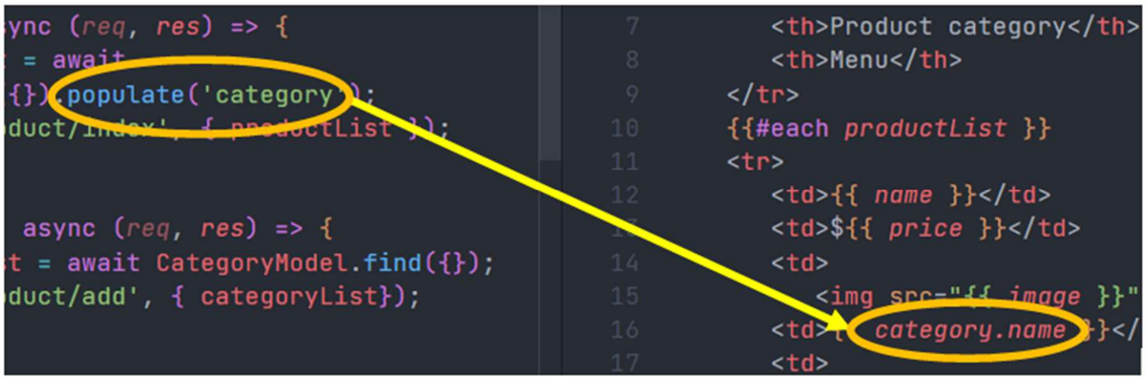


```
get('/add', async (req, res) => {
categoryList = await
yModel.find({});
render('product/add', { categoryList});


post('/add', async (req, res) => {
product = req.body;
it ProductModel.create(product);
```

```
Price: <input type="number" nam
<br><br>
Image: <input type="url" name="
<br><br>
Category:
<select name="category" id="">
  {{#each categoryList }}
  <option value="{{_id}}">
    {{ name }}
  </option>
```

Figure 43 - Add product form (make ***drop-down list*** *to* ***select category****: display category* ***name*** *but add* ***_id*** *into database)*

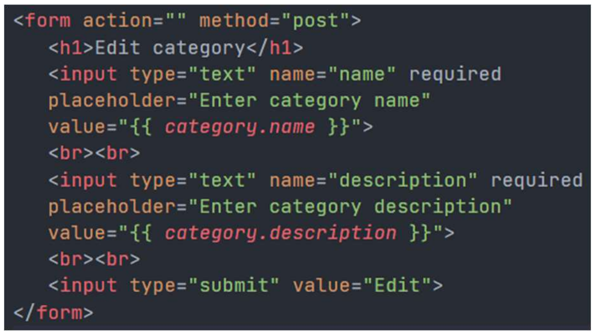


```
ync (req, res) => {
 = await
{}).populate('category');
duct/index', { productList });


 async (req, res) => {
t = await CategoryModel.find({});
duct/add', { categoryList});
```

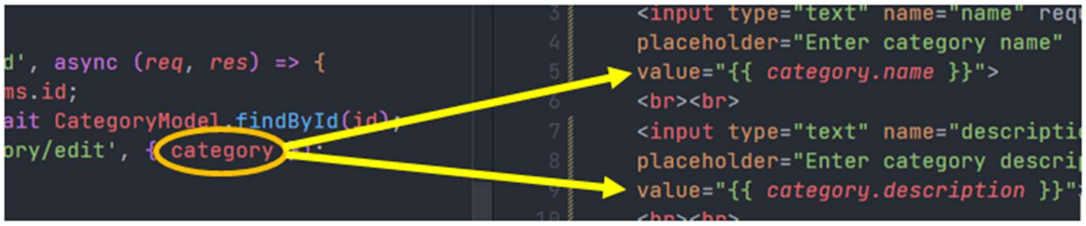
```
      <th>Product category</th>
      <th>Menu</th>
    </tr>
    {{#each productList }}
    <tr>
      <td>{{ name }}</td>
      <td>${{ price }}</td>
      <td>
        <img src="{{ image }}"
      <td>{{ category.name }}</
      <td>
```

Figure 44 - Use ***populate*** *for* ***reference column*** *to* ***display data*** *from* ***reference table\***



```html
<form action="" method="post">
   <h1>Edit category</h1>
   <input type="text" name="name" required
   placeholder="Enter category name"
   value="{{ category.name }}">
   <br><br>
   <input type="text" name="description" required
   placeholder="Enter category description"
   value="{{ category.description }}">
   <br><br>
   <input type="submit" value="Edit">
</form>
```

Figure 45 - Edit category form

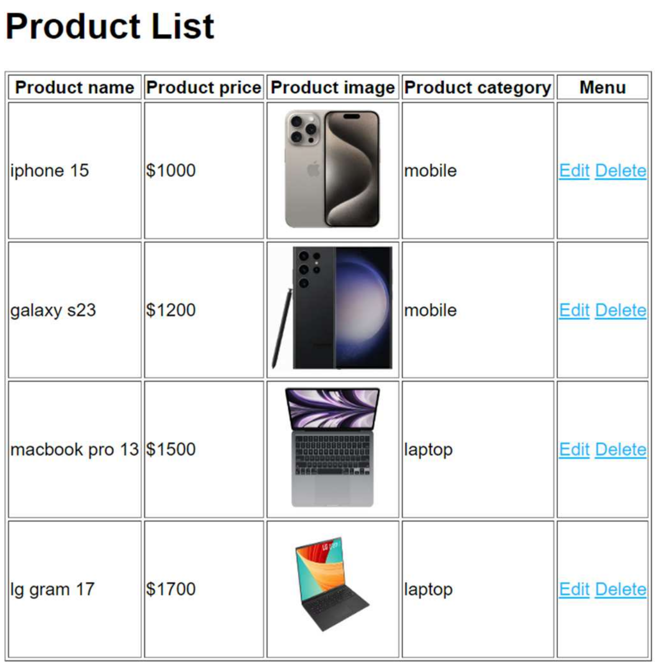


```
d', async (req, res) => {
ms.id;
ait CategoryModel.findById(id);
ory/edit', { category });
```

```
<input type="text" name="name" req
placeholder="Enter category name"
value="{{ category.name }}">
<br><br>
<input type="text" name="descripti
placeholder="Enter category descri
value="{{ category.description }}"
```

9. Test the web app to see the result

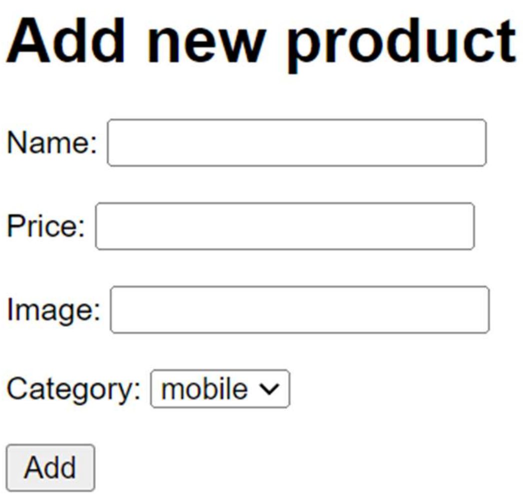
<u>Note:</u> Website interface is quite basic at present due to lack of CSS



Product List

Product name	Product price	Product image	Product category	Menu
iphone 15	$1000		mobile	Edit Delete
galaxy s23	$1200		mobile	Edit Delete
macbook pro 13	$1500		laptop	Edit Delete
lg gram 17	$1700		laptop	Edit Delete

Figure 46 - Product list



Add new product

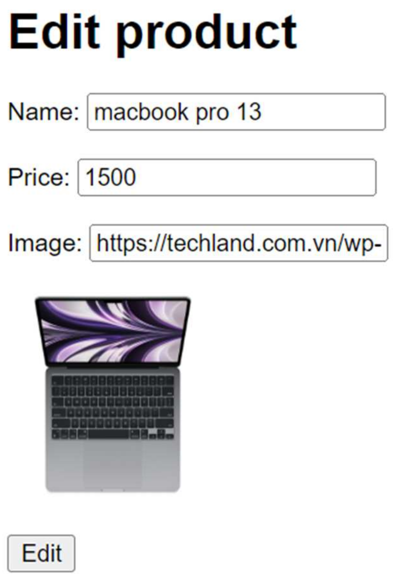
Name:

Price:

Image:

Category: mobile

Figure 47 - Add new product



Edit product

Name: macbook pro 13

Price: 1500

Image: https://techland.com.vn/wp-

Figure 48 - Product edit