**Mongoose Store**

Make a product inventory manager with full CRUD using Mongoose.

**Learning Objectives**

* Full CRUD app in Express with Mongoose

**Prerequisites**

* JavaScript
* Express / Node
* Mongo / Mongoose

**Expected Functionality (MVP)**

**Index Page**

1. Your app should have an index page where
   * all the products are displayed
   * the images link to the product's show page
   * and there should be a link to add a new product.

**Example**[A screenshot of a cell phone

Description automatically generated](https://i.imgur.com/CRJd6Hg.png)

**Show Page**

1. Your show page should display a product with
   * a link back to the products
   * a link to edit the product (goes to the edit page)
   * a delete button that deletes
   * and the number of items remaining in stock.
2. There should be a BUY button. The BUY button will reduce the number of items in stock by 1 each time it's pressed.

**Example**

1. If the quantity of your item is zero, the show page should say 'OUT OF STOCK' instead of saying how many are remaining. (Hint: conditionals in ejs).
2. On the edit page, make sure you can set the quantity to zero if you want so that you can test if this is working.
3. The BUY button should also **not** be rendered if the quantity of the item is zero.

**Example**

**Edit & New Page**

1. These views should render forms and submit to the appropriate routes.

**Redirects**

1. The create route should redirect to the index
2. The delete route should redirect to the index
3. The update route will redirect back to the product's show page.
4. For the Hungry for more? the BUY button will go to a route that redirects back to the product's show page

**Getting Started**

1. Inside the mongoose\_store folder, set up Express with MVC architecture with the appropriate folders for models, views, and controllers.
2. You will need the seven RESTful routes. You can begin with your data-layer and test that everything works with cURL. Don't worry about what the BUY button does or where it goes just yet. Just set up your regular RESTful stuff.
3. You will need to make a Mongoose Schema in a products.js file for your products. The schema should have:
4. name: String,
5. description: String,
6. img: String,
7. price: Number,

qty: Number

1. Set up validations for the price and qty (can't be less than zero) and make the name a required field.
2. Create a model and export it.
3. Make sure you connect to your Mongo server in server.js
4. Make sure your controller can access your model:

const Product = require('./models/products');

1. *NOTE:* For testing purposes, especially for having quick access to those wacky Mongo ids, maybe think about having a route /json that res.sends an index of all the data in json format so that you can copy/paste ids into your url bar or cURL or what-have-you.

**The Buy Button**

After you have your full CRUD app working, it's time to break/extend RESTful conventions according to your own lights. The app needs a buy button. It's up to you to make your own routes to facilitate it.

As mentioned in the expected functionality, if a product is in stock (the qty is above 0), the show page should have a BUY button. If the product is out of stock, it should not have this button.

When the BUY button is pressed, it will make a request to update the qty of the product (decrease it by 1).

**Things to think about:**

* What route should the BUY request go to? Maybe it could go to its own route.
* Since it updates the product, should it go to a PUT route?
* Do you need to send any data through to the route? You will need the id, but that is likely all you'll need.
* Can you edit the qty value just in the route? product.qty -= 1?
* Will you have to product.save() if you do this?

**Seed Data**

You can use these seeds to get some starting data if you so choose.

*HANDY HINT:* Make a route in your products controller /seed/newproducts (you can do that by pasting the code below into your controller), and to seed your database, just visit the route once in your browser.

app.get('/seed', async (req, res) => {

const newProducts =

[

{

name: 'Beans',

description: 'A small pile of beans. Buy more beans for a big pile of beans.',

img: 'https://cdn3.bigcommerce.com/s-a6pgxdjc7w/products/1075/images/967/416130\_\_50605.1467418920.1280.1280.jpg?c=2',

price: 5,

qty: 99

}, {

name: 'Bones',

description: 'It\'s just a bag of bones.',

img: 'http://bluelips.com/prod\_images\_large/bones1.jpg',

price: 25,

qty: 0

}, {

name: 'Bins',

description: 'A stack of colorful bins for your beans and bones.',

img: 'http://www.clipartbest.com/cliparts/9cz/rMM/9czrMMBcE.jpeg',

price: 7000,

qty: 1

}

]

try {

const seedItems = await Product.create(newProducts)

res.send(seedItems)

} catch (err) {

res.send(err.message)

}

})

**Commits**

The order in which you tackle this homework is up to you, but keep in mind that because this is a two-night homework, you haven't learned everything yet! Start with what you know and whenever you get to any of the following milestones, commit your work!

**Milestones to Commit**

**Style Inspiration**

See some previous student's examples for some inspiration for your store. Great homework can make great portfolio pieces too!

**Example 1Example 2**

**Hungry for More?**

**2nd Model**

1. Make another model, this time for a User. The User will have:
2. username: String,

shopping\_cart: Array

1. On the product show page, when a user presses BUY, the product will be added to the User's shopping cart.
2. View the shopping cart on the User's show page. (The User will need only a show page and none of the other routes).