Exam 3, Final Prep

* Initialize a NodeJS project
* Create an ExpressJS HTTP server
* Add EJS to the ExpressJS HTTP server
* Configure the ExpressJS HTTP server to read URL-encoded post data
* Read the data.json file and parse it as JSON to an object called **data**
* GET **/** returns an EJS template with a list of all orders and displays the following:
  + Link to **/neworder** to start a new order
  + Link to **/additem** to add items to an order
  + Link to **/view** to view a list of orders
* GET **/neworder** returns an EJS template with an HTML form containing the following:
  + Text Field: **Customer Name**
  + Text Field: **Customer Address**
  + Submit Button that POSTs to **/neworder**
* POST **/neworder** returns an EJS template. What that template shows depends on the data validation:
  + If the **Customer Name** or **Customer Address** fields were blank, return a message to the user telling them they left one of the fields blank
  + Otherwise, create a new object with the following properties, and push it into the **data** object’s **orders** array, then return a message telling them “Order X was created” (X is the **ordernumber** property below)
    - **ordernumber** is the length of the **data** object’s **order** property
    - **customername** is the provided **Customer Name**
    - **customeraddress** is the provided **Customer Address**
    - **items** is an empty array
    - **subtotal** is 0
    - **tax** is 0
    - **total** is 0
  + Convert the **data** object to a JSON string, and save it to the **data.json** file
  + In either case, add a link back to **/**
* GET **/additem** returns an EJS template with an HTML form containing the following:
  + Text Field: Order Number
  + Text Field: Item Name
  + Text Field: Quantity
  + Text Field: Price
  + Submit Button that POSTs to **/neworder**
* POST /**additem** returns an EJS template. What that template shows depends on the data validation:
  + If the any of the text fields were blank, return a message to the user telling them they left one of the fields blank
  + If the order number is between 0 and the **data** object’s **orders** length minus 1, return a message to the user telling them that order does not exist
  + Otherwise, create a new object with the following properties, and push it into the **items** array of the **orders** element whose index is theprovided **Order Number** in the **data** object.
    - **itemname** is the value of **Item Name**
    - **quantity** is the provided **Quantity**
    - **price** is the provided **Price** rounded to the nearest penny
  + If the order number was provided:
    - Reset the **subtotal**, **tax**, and **total** to 0
    - For each item in the **items** array for the provided order number:
      * Multiply the **price** by the **quantity** and add to **subtotal**
    - Multiply the **subtotal** by 0.06 (6% sales tax) and save it to the **tax**
    - Add **tax** to the **subtotal** and save it to the **total**
  + Convert the **data** object to a JSON string, and save it to the **data.json** file
  + If all form data was valid, return a message to the user saying the item was added successfully
  + In either case, add a link back to **/** and **/additem**
* GET **/view** returns and EJS template whose contents depends on the following:
  + If there is no query parameter provided, return an ‘Unordered List” of all items in the **data** object’s **orders** property with the properties **ordernumber** and **customername**
    - Using EJS, ifthe array is empty, show “No Orders Yet”
  + If the was a query parameter called **order:**
    - If the **order** query parameter is between 0 and the **data** object’s **orders** length minus 1:
      * Return a table showing all of the order’s information, including:
        + Order Number
        + Customer Name
        + Customer Address
        + Every item in the **items** array:

Name

Price

Quantity

* + - * + Subtotal
        + Tax
        + Total
    - Otherwise, return a message to the user “Invalid order number”
  + In any case, add a link back to **/** and **/view**