Table of Contents

Hypothetical Meals

This application is a food inventory system, an alternative solution to spreadsheets.

This product is created by Team 1, the Blubs.

(Teddy, Aninda, David, Lucy)

Developer Guide

This is a developer guide on the Hypothetical Meals inventory system.

Getting Started

First, it's a good idea to get familiar with our stack.

Technologies used:

- Framework: expressjs
- Backend: nodejs
- Database: mongodb
- Frontend: HTML, CSS, Javascript and standard libraries (bootstrap, jquery)

Setup

1. If you have not already done so, make sure you have installed node and npm. See here: https://docs.npmjs.com/getting-started/installing-node
2. If you have not already done so, setup a local version of mongodb or, if you have an existing mongodb database in the cloud that you don't mind using, you may use that too.
3. Clone the source code
4. On a mac: In terminal, navigate to the root directory of the project and then run

```
npm install
```

This will install all the relevant modules you need as indicated in the `package.json` file.

1. Create a file and fill out the relevant values.

```
{
    "development": {
        "MONGO_URI": "<DEVELOPMENT MONGODB URI HERE>",
        "MONGO_OPTIONS": { "db": { "safe": true } }
    },
    "production": {
        "MONGO_URI": "<PRODUCTION MONGODB URI HERE>",
        "MONGO_OPTIONS": { "db": { "safe": true } }
    },
    "email": "<EMAIL HERE>",
    "password":"<PASSWORD HERE>"
}
```

1. Run `npm start` while still in your root directory. Navigate to `http://localhost:3000/` in your browser.

How It Works

The application uses an Model View Controller architecture. Please read the specific sub pages for more details.

The models are: Vendor, Ingredient, User, Token, Cart, Inventory.

The controllers, also known as routes in an expressjs app, correspond to the REST API and are supported for the following root endpoints: vendors, ingredients, users, inventory, files.

The views are simply pug (an HTML templating language used in expressjs apps) files that extend an overarching layout file. The theme is customized in CSS and javascript files.

Models

The schema for each model is also displayed. One reason mongodb was a good database decision was because of how easily it lends itself to representing objects in javascript (essentially just json or dictionaries).

Vendor: An admin can create, read, update and delete (CRUD) a vendor.

Schema:

```
{
  name: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  code: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  contact: {
    type: String,
    required: true,
    trim: true
  },
  location:{
    city:{
      type:String,
      required:true,
      trim: true
    },
    state:{
      type:String,
      required:true,
      trim: true
    }
  },
  catalogue:[{
    ingredient: {
      type: String,
      required: true,
      trim: true
    },
    temp:{
      type: String,
      required: true,
      trim: true
    },
    package:{
      type: String,
      required: true,
      trim: true
    },
    available:{
      type: Number,
      required: true,
      trim: true
    },
    cost:{
      type: Number,
      required: true,
      trim: true
    }

  }],
  history:[{ingredient:String, cost:Number, number:Number}]
}
```

User: CRUD implementation

Schema:

```
{
  email: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  username: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  password: {
    type: String,
    required: true,
  },
  passwordConf: {
    type: String,
    required: false,
  },
  isVerified: {
    type: Boolean,
    default: false
  },
  role: {
    type: String, // "Admin" or "User"
    required: true,
  },
  cart: {
    type: Array
  }
}
```

Ingredient: CRUD implementation

Schema:

```
{
  name: {
    type: String,
    unique: true,
    required: true
  },
  package: {
    type: String,
    enum: ['sack', 'pail', 'drum', 'supersack', 'truckload', 'railcar'],
    required: true,
    trim: true
  },
  temperature: {
    type: String,
    enum: ['frozen', 'refrigerated', 'room temperature'],
    required: true,
    trim: true
  },
  amount: {
    type: Number,
    required: true
  }
}
```

Inventory: A singleton esque model that is a single document in a separate collection in the database. It oversees all of the storage quantities and enforces limitations.

Schema:

```
{
  type:String, //where type is 'Master'
  limits: {
    refrigerated:Number,
    frozen:Number,
    room:Number
  },
  current: {
    refrigerated:Number,
    frozen:Number,
    room:Number
  }
}
```

Cart: The cart a user uses the checkout ingredients.

Schema:

```
{
  ingredient: {
    type: String,
    required: true
  },
  quantity: {
    type: Number,
    required: true
  }
}
```

Token: The token model is used for email verification when a user is created. When an account is created, an email verification token must be created and then appended to the end of the confirmation URL that is sent to the new user account email.

Schema:

```
{
  _userId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },
  token: { type: String, required: true },
  createdAt: { type: Date, required: true, default: Date.now, expires: 43200 }
}
```

Routes

The routes describe the endpoints that the REST API calls.

Users

Renders the main users page which is the login page if no one is logged in, else the profile page

```
GET /users
```

Creates or logs a user in depending on if the user is filling out the create account form or login form.

Body Parameters	
For account creation:	
email	String
password	String
username	String
role	String. "admin" or "user"
For login:	
logemail	String
logpassword	String

```
POST /users
```

Allows user to confirm their account creation. This link is sent in the email verification and will never be called independently.

Query Parameters	
id	String. The token id generated from email verification

```
GET /users/confirmation
```

Allows the user to resend their verification token.

Body Parameters	
email	String. The email of the account for which you want the verification token resent.

```
POST /users/resendToken
```

Display the user profile

```
GET /users/profile
```

Delete a user. Only an admin can do this.

Body Parameters	
email	String. The user account email to delete.

```
POST /users/delete
```

Update a user.

Body Parameters	
email	String. The email of the account to update. This cannot be changed.
username	String. The new username to which you want to update.
password	String. The new password to which you want to update.

```
POST /users/update
```

Returns an object indicating if the current user is or is not an admin.

```
GET /users/isAdmin
```

Returns an object indicating if the current user is or is not logged in.

```
GET /users/isLoggedIn
```

Returns the cart page

```
GET /users/cart
```

Add ingredients to the user's cart

Body Parameters	
ingredient	String. The name of the ingredient
quantity	Number. The quantity of the ingredient.
amount	Number. The price of the ingredient.

```
POST /users/add_to_cart
```

Remove an ingredient from the user's cart

Body Parameters	
ingredient	String. The name of the ingredient.

```
POST /users/remove_ingredient
```

Checkout the cart items. This also updates the inventory status.

Body Parameters	
ingredient	String. The name of the ingredient.
quantity	Number. The quantity of the ingredient.
amount	Number. The price of the ingredient.

Vendors

Get the vendor for a specific code

```
GET /vendors/:code
```

Delete the vendor for a specific code

```
POST /vendors/:code/delete
```

Update the vendor for a specific code

```
POST /vendors/:code/update
```

Create a new vendor

Body Parameters	
name	String. The vendor name.
contact	String. The vendor contact information.
location	String. The vendor location.

```
POST /vendors/new
```

Add ingredients to a vendor for a specific code.

```
POST /vendors/:code/add_ingredient
```

Update an ingredient for a vendor for a specific code.

```
POST /vendors/:code/update_ingredient
```

Place an order from a vendor with a specific code.

```
POST /vendors/:code/order
```

Ingredients

Get ingredient by name

```
GET /ingredients/:name
```

Get ingredient by name and amount

```
GET /ingredients/:name/:amount
```

Delete ingredient by name

```
POST /ingredients/:name/delete
```

Update ingredient by name

```
POST /ingredients/:name/update
```

Create ingredient

Body Parameters	
name	String. Ingredient name.
package	String. Package type.
temperature	String. Temperature.
amount	Number. Amount of the item.

```
POST /ingredients/new
```

Inventory

Update the inventory temperature storage limits.

Body Parameters	
room	Number. The new room temperature limit.
frozen	Number. The new frozen temperature limit.
refrigerated	Number. The new refrigerated temperature limit.

```
POST /inventory/update_limits
```

Deployment

Setup

Requirements:

- node version >= 7.6
- mongodb database

Local setup

1. Download the code
2. Run `npm install` in the root directory.
3. Create a file `env.json` in the root directory. It should look like the following (but with your own relevant values):

```
{
    "development": {
        "MONGO_URI": "<DEVELOPMENT MONGODB URI HERE>",
        "MONGO_OPTIONS": { "db": { "safe": true } }
    },
    "production": {
        "MONGO_URI": "<PRODUCTION MONGODB URI HERE>",
        "MONGO_OPTIONS": { "db": { "safe": true } }
    },
    "email": "<EMAIL HERE>",
    "password":"<PASSWORD HERE>"
}
```

1. Run `node setup.js` to create your default admin user
2. Run `npm start` to start the app locally

Deployment

We used heroku to deploy. Heroku comes with a free mongodb add-oncalled mLab that you will need. Additionally, set the environment variables under the Settings tab, where the Config Variables are located. You will need to set the EMAIL and PASSWORD environment variables. The MONGODB_URI should have already been set when you added the mlab add on.

To actually deploy to heroku, you will need a heroku account and the heroku command line tools. There is already a good tutorial on heroku.
Because heroku also uses git for version control, you can simply push to your heroku remote in order to deploy (ie. `git push heroku master`)