
Table of Contents

| | |
|--------------------|-----------|
| Introduction | 1.1 |
| Developer Guide | 1.2 |
| Getting Started | 1.2.1 |
| How It Works | 1.2.2 |
| Models/Schema | 1.2.2.1 |
| Controller/Routes | 1.2.2.2 |
| Users | 1.2.2.2.1 |
| Vendors | 1.2.2.2.2 |
| Ingredients | 1.2.2.2.3 |
| Inventory | 1.2.2.2.4 |
| Reports | 1.2.2.2.5 |
| Files | 1.2.2.2.6 |
| Formulas | 1.2.2.2.7 |
| Duke Oauth | 1.2.2.2.8 |
| Backups | 1.2.2.2.9 |
| Deployment Guide | 1.3 |
| Backup Admin Guide | 1.4 |

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>",
  "tokenHost": "https://oauth.oit.duke.edu",
  "clientId": "<CLIENT SECRET ID>",
  "clientSecret": "<CLIENT SECRET HERE>",
  "redirectURI": "https://hypothetical-meals-dev.herokuapp.com/duke_oauth/callback"
}
```

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, 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:{
    type: String,
    required: false,
    trim: true
  },
  catalogue:[
    {
      ingredient:{type:mongoose.Schema.Types.ObjectId, ref:'Ingredient'},
      cost:Number
    }
  ],
  history:[{ingredient:String, cost:Number, number:Number}]
}
```

User: CRUD implementation

Schema:

```
{
```

```
email: {
  type: String,
  required: false,
  trim: true
},
username: {
  type: String,
  required: false,
  trim: true,
  unique: true
},
password: {
  type: String,
  required: false,
},
passwordConf: {
  type: String,
  required: false,
},
netid: {
  type: String,
  required: function() {
    return this.isDukePerson ? true : false
  }
},
isDukePerson: {
  type: Boolean,
  default: false
},
role: {
  type: String, // "Admin" or "User" or "Manager"
  default: 'user',
  required: true,
},
cart: [{
  ingredient: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Ingredient',
    required: true
  },
  quantity: {
    type: Number,
    required: true
  },
  vendors: {
    type: Array,
    required: true
  }
}],
report: {
  type: Array
},
production_report: {
```



```
    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
  },
  nativeUnit: {
    type: String,
    required: true,
    trim: true
  },
  unitsPerPackage: {
    type: Number,
    required: true,
    trim: true
  },
  amount: {
    type: Number,
    required: true
  },
  averageCost: {
    type: Number
  },
  space: {
    type: Number,
    required: true
  },
  isIntermediate: {
    type: Boolean,
    required: true
  },
  vendors: [{
```

```
    vendorId: {
      type: String,
      trim: true
    }
  }],
  vendorLots: [{
    vendorID: String,
    lotNumber: String,
    units: Number,
    timestamp: Number
  }]
}
```

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
  }
}
```

Formula: CRUD implementation.

Schema

```
{
  name: {
    type: String,
    unique: true,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  tuples:[{
    index: {
      type: Number,
      required: true
    },
    ingredient: {
      type: String,
      required: true
    },
    ingredientID: {
      type: String,
      required: true
    },
    quantity:{
      type: Number,
      required: true
    }
  }],
  units: {
    type: Number,
    required: true
  }
}
```

Log: The model for the global logging system log.

Schema

```
{
  title: {
    type: String,
    required: true
  },
  description: {
    type: String
  },
  time: {
    type: Date,
    default: Date.now,
    required: true
  },
  entities: {
    type: Array,
    required: true
  },
  initiating_user: {
    type: ObjectId,
  }
}
```

Production: The model for the production report.

Schema:

```
{
  name: {
    type: String,
    required: true,
    unique: true
  },
  product: [
    {
      formulaId: {
        type: mongoose.Schema.Types.ObjectId
      },
      formulaName: String,
      unitsProduced: Number,
      totalCost: Number
    }
  ]
}
```

Spending: The model/singleton for the spending report.

Schema:

```
{
  name: {
    type: String,
    required: true,
    unique: true,
    enum: ['spending', 'production']
  },
  spending: [
    {
      ingredientId: {
        type: mongoose.Schema.Types.ObjectId
      },
      ingredientName: String,
      totalSpent: Number
    }
  ]
}
```

Recall: The model/singleton for the recall report

```
{
  name: {
    type: String,
    required: true,
    unique: true
  },
  lotmap: [
    {
      lotnumber: String,
      productList: [foodProductID: mongoose.Schema.Types.ObjectId];
    }
  ]
}
```

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
```

Edits an order in the cart.


```
GET /users/edit_order/:ingredient/:page?
```

Remove an ingredient from the user's cart

| Body Parameters | |
|-----------------|-------------------------------------|
| ingredient | String. The name of the ingredient. |

```
POST /users/remove_ingredient
```

Provide information about the order that has been edited.

| Body Parameters | |
|-----------------|--|
| ingredient | String. The name of the ingredient. |
| quantities | Array. The quantities of the ingredient. |
| names | Array. The names of the vendors from which the ingredient is being ordered from. |
| codes | Array. The codes of the vendors from which the ingredient is being ordered from. |

```
POST /users/edit_order
```

Checkout the cart and place the order.

```
POST /users/checkout_cart
```

Vendors

Get the list of all vendors

```
GET /home/:page?
```

Get the vendor for a specific code

```
GET /vendors/:code/:page?
```

Get the vendor from object id

```
POST /vendor/id/:vendor_id
```

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. |
| code | String. The vendor freight code. |
| contact | String. The vendor contact information. |
| location | String. The vendor location. |

```
POST /vendors/new
```

Add ingredients to a vendor catalogue for a specific code.

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

Remove ingredients from a vendor catalogue for a specific code

```
GET /vendors/:code/:code/remove_ingredient/:ingredient
```

Update an ingredient in the catalogue 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

| Body Parameters | |
|-----------------|---|
| name | String. New ingredient name. |
| package | String. New package type. Must be a pre-defined choice. |
| temperature | String. New temperature. Must be a pre-defined choice. |
| amount | Number. New amount of the item. |

```
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
```

Associate ingredient with a vendor

```
POST /ingredients/:name/add-vender
```

Add an ingredient order to the cart

```
POST /ingredients/order/add/to/cart
```

Update an ingredient's lot

```
POST /ingredients/:name/updateslot
```

Edit an ingredient's lot amount

```
POST /ingredients/:name/edit-lot/:oldAmount
```

Delete a lot from an ingredient

```
POST /ingredients/:name/delete-lot/:amount
```

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

Reports

Get reports. Returns page with information for all of the reports: spending, production, recall and ingredient freshness.

```
GET /reports/
```

Files

Base route

```
GET /files/
```

Get bulk import documentation

```
GET /files/documentation
```

Upload ingredient CSV

```
POST /files/upload/ingredients
```

Upload formula CSV

```
POST /files/upload/formulas
```

Upload intermediate products

```
POST /files/upload/intermediates
```


Formulas

Renders the main formulas page.

```
GET /formulas/home
```

Renders the page of a specific formula.

```
GET /formulas/:name
```

Creates a new formula.

| Body Parameters | |
|-----------------|--|
| name | String. The name of the new formula. |
| description | String. The description of the new formula. |
| units | Number. The number of units produced by the new formula. |

```
POST /formulas/new
```

Deletes a formula.

| Body Parameters | |
|-----------------|--|
| name | String. The name of the formula to delete. |

```
POST /formulas/:name/delete
```

Updates a formula.

| Body Parameters | |
|-----------------|--|
| name | String. The new name of the formula to update. |
| description | String. The new description of the formula to update. |
| units | Number. The new number of units produced by the formula to update. |

```
POST /formulas/:name/update
```

Deletes a tuple in a formula.

| Body Parameters | |
|-----------------|---|
| name | String. The name of formula whose tuple is being deleted. |
| id | ObjectId. The id of the tuple being deleted. |

```
POST /formulas/:name/delete_tuple
```

Duke Oauth

This route is in charge of redirecting the user to the duke Duke shibboleth for authentication. After the user is redirected, the user logs in and then sent to the redirect URI with the authorization token in the URL. The duke identity API is called to get the user's information and the user is logged in.

Backups

Backups utilizes mongodb-backup and mongodb-restore to create backups and allow for system restores.

Make a backup

```
/backups/makebackup
```

Download a backup tar file using filename

```
/backups/file/:filename
```

Restore system to a backup by id

```
/backups/restore/:id
```

Delete a backup by id

```
/backups/delete/:id
```

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-on](#) called 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`)

Backup

We create mongodb backups and restores using the libraries [mongodb-backup](#) and [mongodb-restore](#), which work with BSON data dumps. These backups are scheduled regularly(daily, weekly, monthly) on chron jobs. These backups are saved on a separate mlab environment with a separate static ip address. Emails are sent out to the hypotheticalfoods458@gmail.com account upon successful or failed backup as well as on a restore.

Email Credentials:

Username: hypotheticalfoods458@gmail.com

Password: hypotheticalfoods458heroku

To view, download or restore to a backup, login to the system with:

IT Person Credentials:

Username: itperson

Password: it_person

Once logged in, you will see a Backups tab in the sidebar. Once you are on the backups page, there will be buttons to download the tar for a backup, as well as to restore to a particular backup. The IT person can also manually generate backups or delete individual backups should they so choose. However, the chron jobs will automatically take care of generating regular backups and cleaning out old backups.

Implementation

As stated before, the libraries used were mongodb-backup and mongodb-restore. A separate backups route is created to make requests to view backups, download backups and make restores. These requests require the IT person permissions, which was added as a new user role. Making backups themselves are not exposed in the REST api as they are taken automatically in the chron job.