MetaAlligators Developer Guide

Introduction

The following document will give a brief description of the primary technologies used in our project, a high-level design of the server-client interface, and our database schema. In order to meet all the requirements outline in the project, we decided to use the MERN stack along with various NPM packages and ReactStrap/MaterialUI to style. Our high-level design involves 3 main stacks: the CSV stack, the User stack, and the Representation stack. The CSV stack handles CSV parsing in terms of committing to the backend, error checking, and data validation. The User stack handles creating/editing/deleting users and restrictions depending on user privileges. And the Representation stack handles representing the data in the backend in a concise and understandable format which also allows the users to make certain changes such as adding/deleting/modifying entries in our database such as SKUs/Ingredients/Product Lines/Manufacturing goals.

Technologies Used

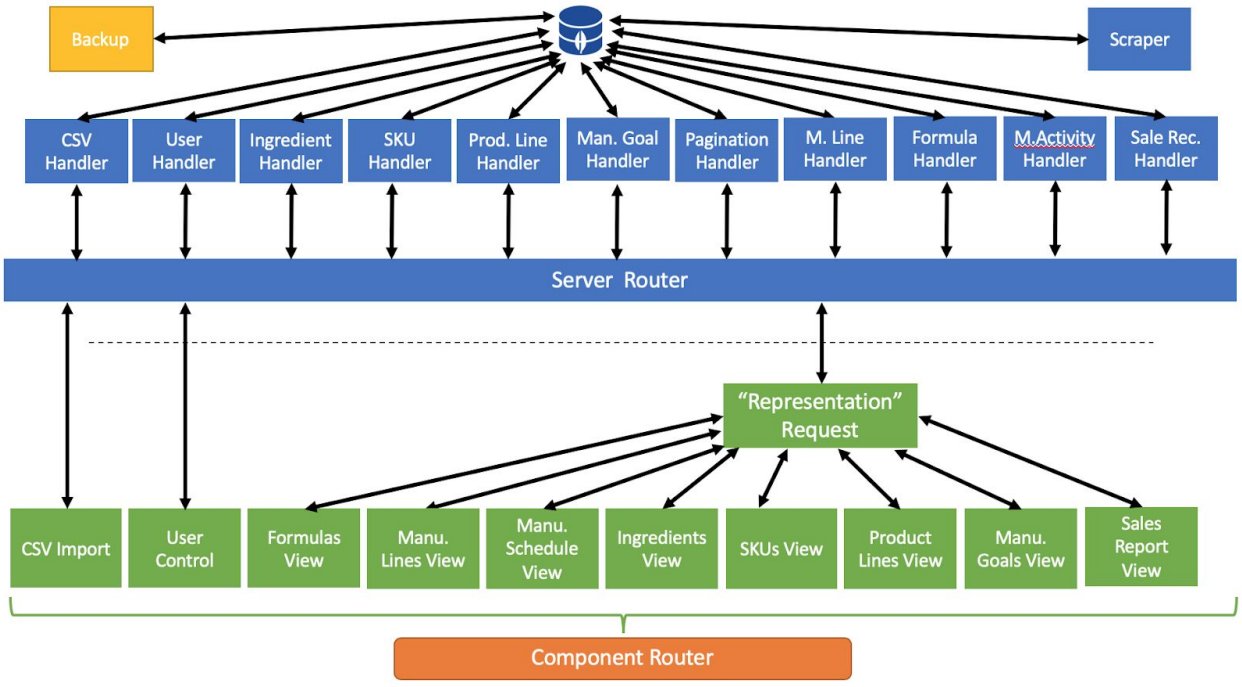
MERN – MERN stands for MongoDB, Express.js, React.js, and Node.js. These four technologies are used for the database, routing, frontend and backend respectively.
ReactStrap – ReactStrap is essentially bootstrap for React.js. It provides a predefined set of stylesheets for components.
MaterialUI – MaterialUI is essentially materialize for React.js. It provides a predefined set of stylesheets for components.
CSVtoJSON - CSVtoJSON is an NPM package that is used to convert CSV files into an array of JSON objects. The package assumes the first row is the set of fields for the JSON objects and then builds each row as a JSON object with those fields.
Redux - Redux is used for login JWT tokens.
Nodemon - Nodemon is an NPM package that is used to allow for faster debugging. The package essentially restarts our development server whenever we make changes to our files. Without Nodemon, we could have to manually restart the server every time we changed anything.
Babel - Babel is an NPM package that is used to convert all Javascript in ES6 to Javascript in ES5. This essentially allows our web server to be backwards compatible which means, although we implemented it used ES6, any browser that only supports ES5 will also be able to access our server.
JSON WebToken - JSON WebTokens are used to keep track of which user is currently logged in. This allows us to manage privileges for different types of users (i.e. admin and non-admin)
Mongoose - Mongoose is an NPM package that is used as an interface with our MongoDB database. Mongoose allows for us to easily and concisely query our database.

Router - Router is a technology that allowed us to route to pages on the front end. It essentially serves the same purpose that Express.js routing does on the back end. It makes sure that the page rendered to the user is the one specified in the URL.
React Vis.JS Timeline - This is an NPM package that is a React conversion of the standard Vis.JS library. It allows dynamically visualizing the schedule of activities for the Manufacturing Schedule.

High-level Design

Database - For our database, we used MongoDB. The database is hosted using a free 500mb service from mLab.
Server - Our server serves as our router. All incoming requests from the front end are initially sent to our Express router and then, based on the string specified for the route, redirected to the appropriate handler.
Backup - The backup module runs independent of the entire system. The script runs on a separate VM that backs-up the data in the Database on a schedule, with all of the backups being stored on that VM.
Scraper - The scraping module runs in the backend of the application, scraping the sales records website on a schedule and pushing the data in the database as Sales Records.
Handlers - Depending on the route determined by our server, one of our handlers will receive and process the request. These handlers contain basic functionality that applies to all collections in our database (creating an entry, updating an entry, etc.) as well as specialized functionality based on the collections associated with the handlers.
CSV Parser - The CSV parsing stack's function is importing a CSV file according to the guidelines specified in *Bulk Import Export Proposal Draft 5*. It implements the single mode of both importing and exporting. This stack also indicates to the user if any errors were found when parsing the file, gives the user the option to accept/reject an import that would cause collisions, and displays an exportable report after an import has been committed to the database.
Submit Request - Submit Request is the interface between front end pages and the server. Whenever one of the four following pages want to communicate with the back end, they submit a Submit Request which deals with the request based on its contents and where it came from. In essence, wraps requests front end requests to be understable on the backend.
Ingredients Page - This page is the user interface for adding/deleting/modifying ingredients. It takes user input and uses Submit Request to send requests to the server.
SKU Page - This page is the user interface for adding/deleting/modifying SKUs. It takes user input and uses Submit Request to send requests to the server. This page takes care of assigning a SKU to a product line.
Manufacturing Goals Page - This page is the user interface for creating/modifying manufacturing goals. It also includes a manufacturing goal calculator that allows you to see an exportable report of quantities of ingredients that are required to meet one's manufacturing goal.
Manufacturing Lines Page - This page is the user interface for editing Manufacturing Lines. Users can export a report for each manufacturing line by choosing a date-span and can edit the

names of manufacturing lines. To change the SKU-mappings to manufacturing lines, users must navigate to the SKUs page and either bulk-edit mappings or edit each SKU.
Manufacturing Schedule Page - This page is only accessible to admins and allows admins to edit currently scheduled activities and place unscheduled activities on the schedule. The interface shows errors for each of the activities as well as a tooltip to understand how to use the page.
Formulas Page - This page is very similar to SKUs and Ingredients in that it shows all of the existing Formulas with options to filter and search. Users can also edit formulas from here, as well as see which SKUs are currently mapped to the formulas.
Product Line Page - This page is the user interface for adding and modifying product lines. This page also handles moving a SKU from one product line to another.
User - This User stack handles logging in, registering new accounts, and managing what certain users can do based on their permissions. This stack also manages encrypting passwords through hashing and salting. There exists a Register New User view and a User Settings view that completes this functionality.
Component Router - The component router deals with routing the user to the correct front end page in the Representation stack. Depending on the URL requested by the user, this component will make sure the correct page is rendered with all the correct functionality. This is essentially the user's interface in terms of which pages to load in the Reference Stack.



Database Schemas

The collections in our database were designed with simplicity and conciseness in mind. We chose Numbers for fields that require some sort of sorting/calculation based on their

numeric value and strings for almost all other fields. We used Schema.Types.ObjectId for certain fields in our schema since it minimized the number of changes we had to make to our collections when processing an update (i.e. updating a product line just updates the name in the product line collection instead of having it do it both in its own collection and in the SKU collection since the SKU will point to the *object id* of the product line). We have calls in our handlers to communicate directly with the database using the Mongoose API which ultimately provides a seamless interface between the user and the database.

SKU Collection
name : String
num : Number
case_upc : Number
unit_upc : Number
unit_size : String
cpc : Number
prod_line : Schema.Types.ObjectId (points to entry in Product Lines collection)
formula : Schema.Types.ObjectId (points to entry in Formula collection)
scale_factor : Number
manu_lines : Schema.Types.ObjectId (points to entry in Manufacturing Lines collection)
manu_rate : Number
comment : String

Ingredient Collection
name : String
num : Number
vendor_info : String
pkg_size : String
pkg_cost : String
sku_count : Number (a count of the number of SKUs that require this ingredient)
comment : String

Manufacturing Goal Collection
name : String
user : String
activities : Array of Schema.Types.ObjectIds (points to Manufacturing Activity)
enabled : Boolean
deadline : Date

Product Line Collection
name : String

User Collection
name : String

password : String
privileges: Array of String
admin_creator : String
isAdmin : Boolean
isNetIDLogin : Boolean
comment : String

Formula Collection
name: String
num: String
ingredients: Array of Schema.Type.ObjectIDs (points to Ingredient)
ingredient_quantities: Array of String
comment: String

Manufacturing Activity Collection
sku: Schema.Type.ObjectID (SKU)
quantity: Number
scheduled: Boolean
start: String
manu_line: Schema.Type.ObjectID (points to Manufacturing Line)
duration: Number
orphaned: Boolean
unscheduled_enabled: Boolean
over_deadline: Boolean
overwritten: Boolean

Manufacturing Line Collection
name: String
short_name: String
comment: String

Sale Record Collection
cust_num: String
cust_name: String
sku_num: String
date: {
    week: Number
    year: Number
}
Sales: Number
ppc: Number

Configuring for Development

Note: You will need an internet connection to run npm commands or clone the git repository

1) Clone the repository into some directory using `git clone`
2) Make sure you are in the highest level directory of the repository and then run `npm install`
3) Change into the directory labeled client using `cd ./client`
4) Run `npm install` again now that you are in the client directory
5) Change back into the highest directory using `cd ..`
6) You can now deploy the server using `npm run start:dev`