Food Forward Tracker Documentation

Introduction:

Our documentation covers the various files and functions that make up Food Forward Tracker. Each section corresponds to a section of code, its component functions, and the functionality that it provides. For example, the items in the Components section of this document refer to each Javascript React component that is defined in the components folder of ffreact-prod. Each subsection then will give you information on each of the functions that make up a component.

Table of Contents:

[Frontend](#_awyky0qjvw91)

[1. Components:](#_5ggwfyuvud6a)

[1.1. CellDialog.js](#_rser9vjxla3z)

[1.2. DatagridDialog.js (Deprecated)](#_ksts1hi2y6dh)

[1.3. Dropdown.js](#_uzenrthvj99s)

[1.4. EntryPage.js](#_ivg2h2x298rr)

[1.5. FormDialog.js](#_xutt4wf8e125)

[1.6. LandingPage.js](#_ghtt2kc87ocy)

[1.7. LoginPage.js](#_mns6xldk4y9a)

[1.8. ModularDatagrid (Deprecated)](#_efmgsnwx8yke)

[1.9. NewModularDatagrid](#_36ptotl747aq)

[1.10. ModularRecipeDatagrid](#_t7b9i36pmzd4)

[1.11. ModularSelect](#_mfbskgjyado3)

[1.12. Navbar.js](#_hpzrij5wzoj)

[1.13. PwResetPage.js (Deprecated)](#_f8r917lmynkv)

[1.14. SearchToolbar.js](#_uv9zwt8ar1ut)

[1.15. UnderConstruction.js](#_d18wog6phk26)

[2. Household:](#_q1wj7296dhw)

[2.1. EditableHouseholdRow.js](#_mfk5c8w5wx35)

[2.2. HouseholdForm.js](#_b07ma9n5lm5a)

[2.3. HouseholdList.js](#_gp6846vuzhl4)

[3. Images:](#_buht7b9n43jj)

[4. Ingredients:](#_xt1a068eq7wp)

[4.1. EditableIngredientRow.js](#_m92vxl2j1f2t)

[4.2. EditableUsageTable.js](#_p2wt93205q0h)

[4.3. IngredientForm.js](#_csxf9ocqwv0y)

[4.4. IngredientList.js](#_smwsn9rpa6jc)

[4.5. IngredientPage.js](#_ccthxpza5uau)

[4.6. IngredientRow.js](#_p3d3u0nqdyzi)

[4.7. IngUsageTable.js](#_8fjayjb12tr3)

[5. Meals:](#_ewnus5orlekx)

[5.1. MealList.js](#_ki503ld3tgbg)

[5.2. MealPlanForm.js](#_plcr6gnorl2d)

[5.3. MealPlanPage.js (Deprecated)](#_ptfqlewq8eun)

[5.4. MealPlanRow (Deprecated)](#_sozwwiknfhg4)

[5.5. EditableMealRow.js (Deprecated)](#_vkhcrv67ynt)

[5.6. MealsPage.js (Deprecated)](#_gcioahmqz5cb)

[5.7. MSRow.js (Deprecated)](#_l78auk8slwfw)

[6. Packaging –](#_dwlsiwul2xcr)

[6.1. EditablePackagingRow.js](#_xy1dtnfayctq)

[6.3. PackagingForm.js](#_6hdu947mxhe8)

[6.4. PackagingList.js](#_hihh3jqie2c9)

[6.5. PackagingPage.js](#_d4ifgkedsint)

[6.6. PackagingRow.js](#_w1q78pqbc2t)

[6.7. PkgUsageTable.js](#_c00r9cx21ufv)

[7. Recipe](#_4xrbstr2fms4)

[7.1. RecipePage.js](#_ju6zf8evjsg3)

[7.2. Recipe.js](#_whziww2kbw5q)

[7.3. RecipeIngForm.js](#_xxws5aivee5q)

[7.4. RecipeInstForm.js](#_4b9bp3imbwce)

[7.5. RecipePkgForm.js](#_104pnr25mbbg)

[8. Reports –](#_ge6ohmai4wrt)

[8.1. Cost Totals:](#_251ji13ibr1q)

[8.2. ClientsReport:](#_c3teiiojtx29)

[8.3. Ingredient Report:](#_9hyteijelpf1)

[8.4. Ingredient Purchasing:](#_jaacf4p4no0j)

[8.5. Meal History:](#_qj1d6avqzlwb)

[8.6. Meal Plan Report:](#_ciof9w45b3pc)

[8.7. Packaging Report:](#_v4nwedtz8zx2)

[8.8. Packaging Purchasing:](#_1zcjkvl79fx)

[8.9. Packaging Returns:](#_a9ugvcz56yf2)

[9. User –](#_e9wxduo06fa)

[9.1. EditableUserRow.js](#_4bx9uxh899y)

[9.2. NewUserPage.js](#_ay849lnp2okb)

[9.3. UserDetailsExample.js](#_5291gwacspu)

[9.4. UserForm.js](#_hdv3hmuir5z7)

[9.5. UserList.js](#_hl64fwwlx8ro)

[9.6. UserModularDatagrid.js](#_6ead7v8zpdmm)

[9.7. UserPage.js](#_xi6f0wsbtbba)

[9.8. UserRow.js](#_pys7htrw4ghh)

[Backend](#_xucsy6t8np1t)

[Database](#_fr7pmqi65aos)

[Deployment](#_usjytchb5p0o)

[● Steps For Deployment](#_r8v9rxhjjzz9)

# Frontend

Uses React and javascript to define the look and feel of the site.

## Components:

### CellDialog.js

* Defines a reusable MUI Datagrid cell that displays a Button that opens a given component in a MUI Dialog popup
* Takes:
  + buttonText - Text to display in the button
  + dialogTitle - The title to display at the top of the dialog popup
  + component - The component to be displayed in the dialog popup
* Returns:
  + A reusable component that can be rendered in a MUI Datagrid cell that allows a user to open a given component in a popup window.
* Used in:
  + HouseholdPage.js
  + IngredientPage.js
  + PackagingPage.js

### DatagridDialog.js (Deprecated)

* Defines a reusable component that opens a given MUI Datagrid table in a MUI Dialog popup. Can be used to view and edit nested lists of data.
* Takes:
  + open - Boolean React state variable that determines whether the dialog popup should be open or not
  + setOpen - Setter function that sets the open state of this dialog popup
  + DataGridComponent - The MUI Datagrid to display in the dialog popup
  + setID - Function that allows the MUI Datagrid passed to the dialog popup to alter row data in the parent MUI Datagrid.
* Returns:
  + A MUI Dialog component that displays an editable MUI Datagrid table in a MUI Dialog popup.
* Previously used in:
  + Recipe.js

### Dropdown.js

* Defines a reusable dropdown select component that provides a menu of clickable buttons.
* Takes:
  + trigger - Button element from parent component that will be used to open the dropdown menu
  + menu - Array of button elements with onClick functions defined for them to be displayed as the dropdown options.
* Returns:
  + An unordered list of button elements with onClick functions as defined by the menu prop.
* Used in:
  + Navbar.js
  + LandingPage.js
  + MealsPage.js (Deprecated)
  + InventoryPage.js (Deprecated)
  + ReportsPage.js (Deprecated)

### EntryPage.js

* Defines the Entry Page to be shown when a user enters the site at the base URL. Also handles the user authentication cookie to determine whether a user should be re-logged-in on entry.
* Takes:
  + handlePageClick - Function that redirects users to a given page. (Found in App.js)
  + setLoginStae - Setter function to set the login state in App.js for the given user.
* Returns:
  + Entry Page HTML element

### FormDialog.js

* Defines a Mui Dialog popup component that displays a given Form component. Can use an addEntry function to perform the adding of any entered form data.
* Takes:
  + open - Boolean variable that determines whether or not the dialog popup should be displayed.
  + setOpen - Setter function that can close the dialog when necessary.
  + AddFormComponent - Form component that will be displayed in the dialog popup.
  + addFormProps - Any props that should be passed to the AddFormComponent when it is rendered in the dialog popup.
  + latestKey - An optional prop that can provide an ID value to added entries when a new ID cannot be assigned by other means.
* Returns:
  + Dialog Component to be rendered by any other component.
* Used In:
  + NewModularDatagrid.js
  + ModularRecipeDatagrid.js

### LandingPage.js

* Defines the Landing Page to be shown to users after entering through the entry page, or after clicking the Food Forward Logo in the navbar. This page is labeled /home in the URL routing.
* Takes:
  + handlePageClick - function that redirects the user to a given page.
* Returns:
  + Landing Page HTML element

### LoginPage.js

* Defines the Login Page that allows users to login to the site
* Takes:
  + handlePageClick - Function that redirects users to a given page (Found in App.js)
  + setLoginState - Setter function that sets the login state of a user based on a login attempt.
* Returns
  + Login Page HTML element

### ModularDatagrid (Deprecated)

* Defines a MUI Datagrid component that accepts given column definitions, fetches row data from the DB, allows edit/deletion of any row data with confirmation prompts, and provides notifications when saving to the DB succeeds or fails. (Replaced by NewModularDatagrid, which provides more features.)
* Takes:
  + apiEndpoint — The name of the api endpoint that requests should be sent to
  + keyFieldName — The name of the key or ID field of the given row data. Allows indexing on keys/ID’s of row data and requesting specific entries of data in api requests.
  + columns — Array of MUI datagrid column definitions that define the structure of the table data.
* Returns:
  + A reusable MUI datagrid that can be used on any page accessing any data.

### NewModularDatagrid

* Updated version of ModularDatagrid that includes search functionality and Add Form popups, and quality of life features such as a selectable IP for API requests.
* Takes:
  + apiIP — Optional IP address to replace 4.236.185.213 (Useful for testing on local django API using localhost as the IP)
  + apiEndpoint — The API endpoint that the component will send requests to (Example: ingredient-inventory, ing-purchase-report, mealrecipes)
  + keyFieldName — The name of the key or ID field of the given row data. Allows indexing on keys/ID’s of row data and requesting specific entries of data in api requests.
  + AddFormComponent — React component to display in the ‘Add Entry’ dialog popup. Should be a form that allows users to create an entry of the type shown in the Datagrid.
  + addFormProps — The props to pass to the AddFormComponent when it is rendered in the dialog popup.
  + entryName — The name to use in the ‘Add Entry’ button in the format ‘Add {entryName}’
  + columns — Array of MUI datagrid column definitions that define the structure of the table data.
  + columnGroupingModel — A MUI datagrid column grouping definition that creates labels above groups of columns. Can be seen on the Clients page.
  + searchField — The field name of the column that should be searched on. (Example ‘ingredient\_name’ would search by ingredient name on Ingredients)
  + searchLabel — The label for the given searchField that will be shown in the search bar prompt (Example: ‘Ingredient Names’ would display ‘Search Ingredient Names’ in the search prompt)
* Returns:
  + A reusable MUI datagrid that can be used on any page accessing any data. Allows for searching and better debugging.
* Used in:
  + Clients
  + Ingredients
  + Packaging
  + Meals
  + Administration

### ModularRecipeDatagrid

* Fork of ModularDatagrid, used in Recipe.js. All changes are saved to a javascript object, rather than being sent to the DB. In Recipe.js it is used to edit and add to nested lists of Recipe data like recipe ingredients. Any time a Datagrid should update data somewhere other than the DB, a version of ModularRecipeDatagrid would be useful.
* Takes:
  + apiEndpoint — The name of the api endpoint that requests should be sent to
  + keyFieldName — The name of the key or ID field of the given row data. Allows indexing on keys/ID’s of row data and requesting specific entries of data in api requests.
  + AddFormComponent — React component to display in the ‘Add Entry’ dialog popup. Should be a form that allows users to create an entry of the type shown in the Datagrid.
  + addFormProps — The props to pass to the AddFormComponent when it is rendered in the dialog popup.
  + columns — Array of MUI datagrid column definitions that define the structure of the table data.
  + rows — Array of JS objects, usually from the DB, with the actual data to be displayed in the Datagrid. Keys and data types must match the given column definitions.
  + setRows — Callback function that is used to make any updates to data. In Recipe.js this is the setter function for the parent Recipe component’s data state. Can use different functionality to handle different cases.
  + entryName — The name to use in the ‘Add Entry’ button in the format ‘Add {entryName}’

### ModularSelect

* Reusable React Autocomplete component wrapped in a MUI-cell-compatible component. Creates a dropdown select box which accepts new items as well. Uses MUI datagrid api to update Datagrid data, but can be used in add forms or elsewhere.
* Takes:
  + id — optional id of the row being edited in a MUI datagrid. Allows the datagrid api to make any changes to the table data.
  + field — optional field name of the cell being edited in a MUI datagrid.
  + value — optional initial value to be displayed in the cell before changes are made.
  + options — Array of JS objects that is used to get options to display in the dropdown.
  + searchField — The field used to retrieve the values from the options prop that should be displayed in the dropdown. (Example: ‘ingredient\_name’ will populate the dropdown with ingredient names from an array of Ingredient objects.
  + fieldName — The field name of the parent Datagrid column that should be updated with the selected value in the dropdown.
  + required — Optional boolean value that determines whether this input field should be considered required in a form.
  + onChange — Optional handleFormChange callback function that is used to update form data when ModularSelect is used in a form.
* Returns:
  + An auto-completing, searchable, dropdown that can also accept new values. It can be used in a MUI datagrid cell or in a form.
* Used in:
  + IngredientPage.js
  + IngredientForm.js
  + PackagingPage.js
  + PackagingForm.js
  + Recipe.js
  + RecipeIngForm.js
  + RecipePkgForm.js

### Navbar.js

* The navbar that is displayed at the top of the site. Contains links to all other pages.
* Takes:
  + loginState — The current login state of the user. Contains username, isAuthenticated, and isAdmin. Used to determine whether ‘login’ and ‘signup’ should be shown or if ‘my account’ and ‘sign out’ should be shown.
  + handlePageClick — Callback function that redirects the user to a given page.
* Returns:
  + A navbar component with various buttons and dropdowns for each page of the site.

### PwResetPage.js (Deprecated)

* Page that handles resetting user passwords. Not used currently and not in fully-working condition.
* Takes:
  + handlePageClick — Callback function that redirects users to a given page.
* Returns:
  + A page with inputs for a username and a new password.

### SearchToolbar.js

* MUI Datagrid toolbar definition that is passed to a datagrid. Adds a search bar to the toolbar of a parent Datagrid component with a label and handles searching Datagrid data.
* Takes:
  + setFilterModel — callback function from a parent Datagrid component that sets the filter model of that Datagrid.
  + searchField — The field name of the column that should be searched on. (Example ‘ingredient\_name’ would search by ingredient name on Ingredients)
  + searchLabel — The label for the given searchField that will be shown in the search bar prompt (Example: ‘Ingredient Names’ would display ‘Search Ingredient Names’ in the search prompt)
* Returns:
  + MUI toolbar definition. Can only be used in a MUI Datagrid when passed as a prop.

### UnderConstruction.js

* A simple page that displays an ‘Under Construction’ message and image. Can be used as a placeholder while new pages are being developed, or when updates are being made.
* Takes:
  + None
* Returns:
  + Simple component with text and an image that lets users know a page is under construction

## Household:

### EditableHouseholdRow.js

* Handles the dynamic editing of rows in the household table on the clients page

### 2.2. HouseholdForm.js

* Creates the popup Add Client Form. Pulls data from households to populate dropdown menu data. Page will refresh upon closing the window to update the data on the calling page to include new information

### 2.3. HouseholdList.js

* Original page created to call all household talbot data. It does not use DataGrid

2.4. HouseholdModularDatagrid.js

* Defines a MUI Datagrid component that accepts given column definitions, fetches household row data from the DB, allows edit/deletion of any row data, and provides notifications when saving to the DB succeeds or fails. (Replaced by NewModularDatagrid, which provides more features.)

2.5. HouseholdPage.js

* Uses household table to return the household page view. This page displays all the information in the household table and allows users to add new entries via the HouseholdForm.

2.6. HouseholdRow.js

* Original page created to enable dynamically editing rows for HouseholdList.

2.7. KitsList.js

* Original page created to call all kits list talbot data. It does not use DataGrid.

## Images:

Images can be stored here in order to be accessible to the application code whether the database is accessible or not.

## Ingredients:

## EditableIngredientRow.js

* Handles the dynamic editing of rows in the Ingredients table on the Ingredients Page.
* Used in: IngredientsList.js

### EditableUsageTable.js

* Handles the adding, editing and deleting of entries in the ingredient\_usage table.
* Used in: EditableIngredientsRow.js

IngredientPage.js

### IngredientForm.js

* Creates the popup Add Ingredient Form. Pulls data from Ingredients and Suppliers to populate dropdown menu data.   
  The page should refresh upon closing the window to update the data on the calling page to include the new information.
* Used in: IngredientList.js  
   IngredientPage.js

### IngredientList.js

* Original page created to call all Ingredients talbot data and related supplier and usage data. It does not use DataGrid
* Used in: Apps.js

### IngredientPage.js

* Uses Ingredients table, Ingredient\_usage table and the Suppliers table to return the main Ingredients Page view. This page displays all the information in the ingredients table and allows users to add new entries via the IngredientForm.
* Used in: Apps.js

InventoryPage.js

### IngredientRow.js

* Original page created to enable dynamically editing rows for IngredientsList.
* Used in: IngredientList.js

### IngUsageTable.js

* Calls the Ingredient\_Usage table to find all instances where this item was used, removing it from the running total of available items.
* Used in: IngredientPage,js

## Meals:

### MealList.js

* Page with a Datagrid of planned meals. Uses NewModularDatagrid to allow adding/editing/deleting/searching planned meals with dropdowns to select meals and snacks.

### MealPlanForm.js

* Form that allows users to enter new planned meal information. Used with NewModularDatagrid in MealList.js to create the Meal Plan page.
* Takes:
  + recipeList — List of recipes that are used to extract meal and snack names and r\_nums to populate dropdowns. Received from the API in MealList.js.
  + addEntry — Callback function that is used to actually perform the addition of a new entry. Passed by NewModularDatagrid when used as an AddFormComponent.

### MealPlanPage.js (Deprecated)

* Old Meal Plan page using ModularDatagrid. Allows editing/deleting planned meals but does not support searching or adding.

### MealPlanRow (Deprecated)

* Row component that was originally rendered in the HTML Table implementation of the Meal Plan page (older version of MealPlanPage.js)
* Takes:
  + thisKey — The key of this row in the mealList array in the original MealPlanPage.js. Used to update this specific row in the original recipeList.
  + meal — The planned meal object which should be displayed in this row.
  + recipeList — List of recipes that are used to extract meal and snack names to display in the row. Received from the API in the original MealPlanPage.js.
  + deleteMeal — Callback function that is used to delete this row from the mealList array of the parent component.
  + handleEditClick — Callback function that was used in the original MealPlanPage.js to switch this row to edit mode. Would re-render the row as an EditableMealRow component.

### EditableMealRow.js (Deprecated)

* Editable row component that was original rendered in the HTML Table implementation of the Meal Plan Page (older version of MealPlanPage.js)
* Takes:
  + thisKey — The key of this row in the mealList array in the original MealPlanPage.js. Used to update this specific row in the original recipeList.
  + editFormData — The state variable that contains the planned meal object that is currently being edited. This object contains what will be displayed and editable in the row.
  + recipeList — List of recipes that are used to extract meal and snack names and r\_nums to populate dropdowns. Received from the API in the original MealPlanPage.js.
  + updateMeal — Callback function that is used to perform the row update in the original MealPlanPage.js.
  + handleEditFormChange — Callback function that is used to update the editFormData with any changes before the row update is actually performed.
  + handleCancelClick — Callback function that cancels edit mode for this row and sets it back to view mode.

### MealsPage.js (Deprecated)

* A simple page that displays a dropdown menu that can be used to navigate to either the Meal Plans page and the Recipes page.
* Takes:
  + handlePageClick — Function that redirects users to a given page. (Found in App.js)

### MSRow.js (Deprecated)

* An alternative MealPlanRow that was originally written to test how we could display whether a meal or snack was a meal or snack using the m\_s flag on recipe objects. Would have been used in the original HTML Table implementation of MealPlanPage.js. Not to be used anywhere.

## Packaging –

### EditablePackagingRow.js

* Handles the dynamic editing of rows in the Packaging table on the Ingredients Page.

### EditablePkgUsageTable.js

* Handles the adding, editing and deleting of entries in the packaging\_usage table.

### PackagingForm.js

* Creates the popup Add Packaging Form. Pulls data from Packaging and Suppliers to populate dropdown menu data.   
  The page should refresh upon closing the window to update the data on the calling page to include the new information.

### PackagingList.js

* Original page created to call all Packaging talbot data and related supplier and usage data. It does not use DataGrid

### PackagingPage.js

* Uses Packaging table, packaging\_usage table and the Suppliers table to return the main packaging Page view. This page displays all the information in the Packaging table and allows users to add new entries via the PackagingForm.

### PackagingRow.js

* Original page created to enable dynamically editing rows for PackagingList.

### PkgUsageTable.js

* Calls the packaging\_Usage table to find all instances where this item was used, removing it from the running total of available items.

## Recipe

### RecipePage.js

* Base page for Recipes that defines a reworked version of ModularDatagrid which supports adding and editing Recipes.
* Takes:
  + updateDone — Boolean value that is only passed by Recipe.js when the add/update request succeeds and a Recipe is added or updated successfully. Used to open the “Changes Saved” popup.

### Recipe.js

* The Recipe detail page that is used to view, add, or edit recipes. Utilizes ModularRecipeDatagrid for nested lists like Recipe Ingredients, Packaging, and Stations
* Takes:
  + recipeData — The information for a recipe returned from the API meal-recipe-view. Used and updated by the various fields on the Recipe detail page.
  + ingredientOptions — List of ingredient objects as returned from the ingredient-inventory API view. Used to get ingredient names to populate ingredient dropdowns.
  + packagingOptions — List of packaging objects as returned from the packaging-inventory API view. Used to get package types to populate packaging dropdowns.
  + setCurrPage — Callback function from RecipePage.js that redirects the user back to the base Recipes page so they may see their newly edited/added recipe in the list.
  + isAdding — Boolean flag that determines whether the Recipe detail page should be used to add or edit a recipe. Changes the behavior of the api request functions.

### RecipeIngForm.js

* Form that is used to input data for new Recipe Ingredients. Used in Recipe.js to add Recipe Ingredients.
* Takes:
  + addEntry — Callback function that performs the adding of new Recipe Ingredients to the parent component.
  + handleClose — Callback function that sets the ‘open’ state of the form to false in order to close the form.
  + Ingredients — List of ingredient objects as returned from the ingredient-inventory API view. Used to get ingredient names to populate ingredient dropdowns.

### RecipeInstForm.js

* Form that is used to input data for new Recipe Stations. Used in Recipe.js to add Recipe Stations. (Recipe Instructions are now describing Recipe Stations, hence the name)
* Takes:
  + addEntry — Callback function that performs the adding of new Recipe Ingredients to the parent component.
  + handleClose — Callback function that sets the ‘open’ state of the form to false in order to close the form.

### RecipePkgForm.js

* Form that is used to input data for new Recipe Packaging. Used in Recipe.js to add Recipe Ingredients.
* Takes:
  + addEntry — Callback function that performs the adding of new Recipe Ingredients to the parent component.
  + handleClose — Callback function that sets the ‘open’ state of the form to false in order to close the form.
  + packaging — List of packaging objects as returned from the packaging-inventory API view. Used to get package types to populate ingredient dropdowns.

## Reports –

### Cost Totals:

* The Cost Totals report requires the user to select a date range before pulling a list of expenditure for either ingredients or packaging within that range.

### ClientsReport:

* This report shows all client household information in one easily printable location

### Ingredient Report:

* This report displays all the ingredients currently on hand, including their costs and storage requirements.

### Ingredient Purchasing:

* Row

### Meal History:

* Row

### Meal Plan Report:

* Row

### Packaging Report:

* This report displays all the packaging information currently on hand, including their costs and packaging type.

### Packaging Purchasing:

* This report requires the user to select a date range before pulling a list of packaging supplies that are in inventory and what package supply is used for each meal and snack as well as the cost.

### Packaging Returns:

* This report displays the in/out information for package supplies sent out to each client household. It requires the user to select a date range before pulling a list of packaging supplies that are in and out of inventory.

## User –

### EditableUserRow.js

* Handles the dynamic editing of rows in the User table on the user Page.

### NewUserPage.js

* Handles the adding, editing and deleting of entries in the user table.

### UserDetailsExample.js

* Specifies the different type of users that are created. Only accessed by Admin user

### UserForm.js

* Creates the popup Add User Form. Pulls data from User to populate dropdown user data.   
  The page should refresh upon closing the window to update the data on the calling page to include the new information.

### UserList.js

* Uses user table to return the main user Page view. This page displays all the information in the ingredients table and allows users to add new entries via the UserForm.

### UserModularDatagrid.js

* Original page created to enable dynamically editing rows for UserList.

### UserPage.js

* Calls the User table to find all instances where this item was used, removing it from the running total of available items.

### UserRow.js

* Original page created to enable dynamically editing rows for UserList

# Backend

Uses Django and python to create the underlying structures connecting the MySQL database to the site.

Models: Django uses models as the definitive source of information about your data. It contains the essential fields and behaviors of the data you’re storing. Generally, each model maps to a single database table. These models provide an automatically generated database-access API.

Views: A view is a Python function that takes a web request and returns a web response.

# Database

All data for the Food Forward Tracker is housed on an Amazon Web Services (AWS) database listed under the name: **foodforwardserver**, type: **Azure Database for MySQL flexible server**.

Microsoft has guides for connecting and troubleshooting these servers [here](https://learn.microsoft.com/en-us/azure/mysql/flexible-server/overview).

To access the database using MySQLWorkbench or another method of choice, remember to first whitelist your current public IP address under **Networking**.

Tables

auth\_group, auth\_group\_permissions, auth\_permission, auth\_user,

auth\_user\_groups and auth\_user\_user\_permissions:

These standard tables manage different levels of group and user authorization and permissions.

django\_admin\_log: Can be used to log and review database changes.

django\_content\_type: Table listing all data models and their corresponding numbers.

django\_migrations: Logs data migrations.

django\_session: Can be configured to log session information.

hh\_allergies: Table containing a list of allergies.

hh\_kits: Created to track which individual meal deliveries go to which  
 households to enable tracking for reusable packaging returns.

hh\_meal\_plans: Uses Foreign Keys to coordinates the meal\_id from meal\_plans

and meal\_hh\_id from households.

households: Each entry represents one client household with contact details,   
 allergies etc.

ingredient\_usages: This tracks ingredient usages to so used ingredients can be  
 removed from the database.

ingredients: Table listing every intake of ingredients. Items should be entered on  
 new rows even if other rows with the same ingredient already exist.

kit\_packaging: Takes delivery id numbers from kits and matches them to the  
 packaging type to track the number of each type of packaging  
 going into each kit.

kits: Assigns each delivery kit an id number, stores the date it was sent  
 out and to which client household it was sent to.

meal\_packs: Unused. Created to track the process of packing meals.

meal\_plans: Table storing which meals and snacks are sent out on which dates.

packaging: Table tracking the packaging, much like Ingredients it should have  
 new entries made for each incoming purchase.

packaging\_usages: Created to track outgoing reusable packaging.

recipe\_allergies: Unused. Recipes which contain allergens can be entered here.

recipe\_diets: Unused. Built to track which recipes adhere to special diets.

recipe\_ingredients: Each row represents one ingredient that goes into a recipe. How  
 much of it goes into one serving sized portion and may contain  
 instructions for preparing it.

recipe\_instructions: Step by step instructions for each recipe are stored in this table,  
 marked by recipe id and step number for reference.

recipe\_packaging: Tracks the packaging needed for every recipe by type and number  
 as well as a description of what each item will contain.

recipes: This table contains one entry for each recipe entered into the  
 database along with the image file and recipe card path.

station\_ingredients: Unused. Created to plan ingredients by station for recipe prep.

station\_packaging: Unused. Created to plan packaging by station for recipe prep.

stations: Unused. Created to list stations needed for each recipe prep.

supplier: Lists all suppliers and their contact details.

users: Table to track all registered users of the app.

Data

Primary Key:

All tables have one Primary Key, usually the first column, and usually an automatically incrementing number. This guarantees all rows have one unique value for reference.

Foreign Keys:

Most tables have one or more Foreign Keys; these allow tables to reference one another by linking one column in each table and may reference other values.

# Deployment

## Steps For Deployment

* 1. Ensure all changes have been committed pushed to the git
  2. Once changes are in the git, connect to the Azure virtual server using an SSH client.
  3. Navigate to the Apache HTML root by typing into the SSH terminal: cd /var/www/html
  4. Pull any new updates from the git to the server by typing: sudo git pull
  5. Now, back on your local machine, open a terminal in the ffreact-prod folder.
  6. Once the local terminal is running in your ffreact-prod folder, you can build the front-end by typing: npm run build.
  7. Wait until the build is completed, and then open WinSCP and connect to the virtual server.
  8. Navigate again through the WinSCP file system to the /var/www/html directory.
  9. On your local machine, copy all files from the ‘build’ folder in your local ffreact-prod folder and paste them in the /var/www/html directory in WinSCP.
  10. When a popup asks if you would like to overwrite some files, double check that the files are being copied to the base directory at /var/www/html
  11. If so, click yes to all to allow WinSCP to copy all files to the server and overwrite any duplicates.
  12. Now in your SSH client, restart the Apache service by typing: sudo systemctl restart apache2
  13. Once this is complete, all of your changes have been deployed to the server!
  14. **Extra Step**: **Confirming Deployment**
      1. Confirm that these changes have been deployed by navigating to the site at 4.236.185.213
      2. Once at the site’s root url, on the entry page with the “Welcome to Food Forward” press CTRL + SHIFT + R
      3. This will hard refresh the page, clearing the cache for the Food Forward Tracker site and show you the newest content that Apache has available.
      4. If you don’t see your changes, try restarting the apache service or going through these steps again.

## Running the Front-End Code Locally

* 1. Get set up with NPM:
     1. Ensure Node is installed and npm is available to use
     2. Navigate to the ffreact-prod directory
     3. Type: npm install
     4. This will install all necessary packages using the package.json file
  2. In the ffreact-prod directory Run: npm start
  3. The site should open automatically but if not, you can find it at localhost:8000

## Running the Backend Code Locally

* 1. Get set up with python env:
     1. Choose a location for your virtual environment folder:
        + Placing it inside of the fftracker repository folder will cause it to be tracked in git. We would recommend keeping the virtual environment folder outside of your local repository folder.
     2. Type this command to create the virtual environment: python -m venv /path/to/new/virtual/environment
     3. Copy and paste the requirements.txt file from the root of your local git repository into the root of your newly created virtual env folder.
     4. Now in your terminal, navigate into your virtual environment folder
     5. Once in your virtual env folder, type: pip install -r requirements.txt
     6. This will install all of the necessary python packages to run a local instance of the Django API.
  2. Open a terminal and change folder to navigate into the ffdjango\fftracker folder
     1. Run python manager.py runserver
  3. Now once the terminal window displays “Starting development server at http:127.0.0.1:8000” you can be sure that you are running local instance of the django API.

## Testing Local API with Local React:

* 1. Follow the previous steps to get the local React instance running as well as the local Django API instance.
  2. In the front-end page you are trying to test with your local Django API instance, find and replace all instances of the server’s ip ‘4.236.185.213’ with ‘localhost.’
  3. This will send all requests from the front end to your local django API instance.

Support Resources

React: <https://react.dev/>

Django: <https://www.djangoproject.com/>

MySQL: <https://www.mysql.com/>