

Project development Phase

Date	4 November 2023
Team ID	NM2023TMID02112
Project Name	Food Tracking System
Maximum Marks	2 Marks

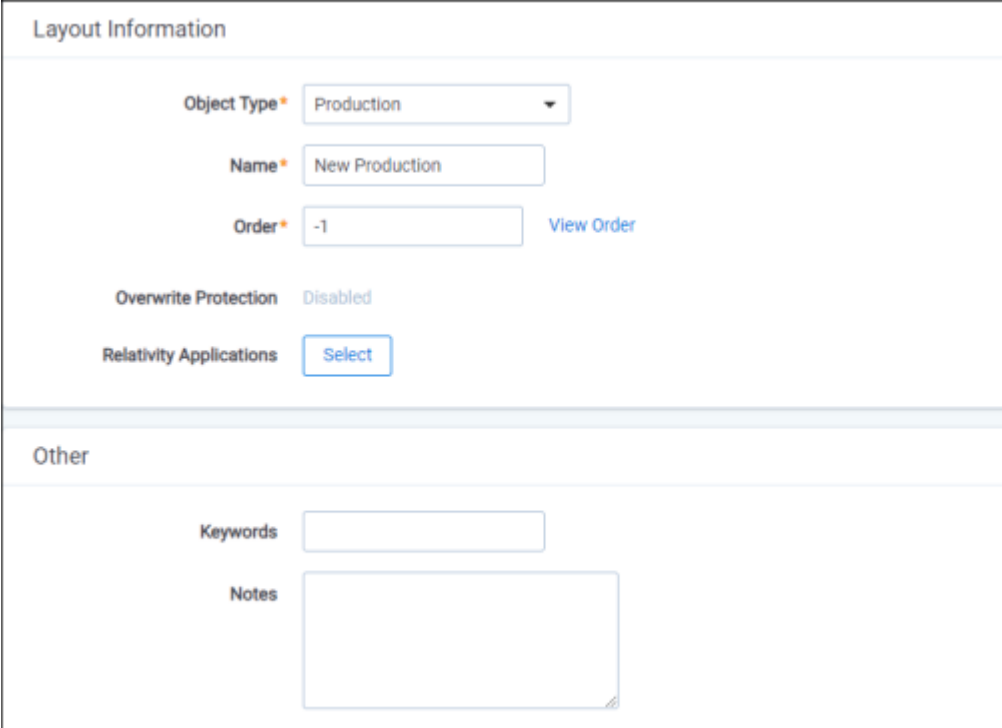
Code Layout

Layouts are web-based coding forms that give you the ability to view and edit document fields. You can use layouts to develop workflows specific to the needs of your case. You can develop layouts that contain only the fields required to complete specific review tasks, making the review process clean and intuitive.

Creating and editing layouts

To create a new layout or edit an existing one:

1. Navigate to the **Layouts** tab.
2. Click **New Layout**. To edit a layout, click the **Edit** link next to the layout's name. The Layout Information details layout opens.
3. Complete the required fields on the Layout Information layout. An orange bar to the right of the field name indicates that it is required. See [Layout Information fields](#)
4. Click **Save**.
5. From the Layout console, click **Build Layout** to open the layout builder. See [Using the layout builder](#).

A screenshot of a web form titled "Layout Information". The form is divided into two main sections. The top section contains several fields: "Object Type" is a dropdown menu with "Production" selected; "Name" is a text input field containing "New Production"; "Order" is a text input field containing "-1", with a "View Order" link to its right; "Overwrite Protection" is a toggle switch currently set to "Disabled"; and "Relativity Applications" is a button labeled "Select". The bottom section, separated by a horizontal line, is titled "Other" and contains two fields: "Keywords" is a text input field, and "Notes" is a larger text area with a small icon in the bottom right corner.

```

1 class PrescribedHeating(TendencyComponent):
2
3     input_properties = {
4         'latitude': {
5             'dims': ['*'],
6             'units': 'degrees_N',
7         },
8         'longitude': {
9             'dims': ['*'],
10            'units': 'degrees_E',
11        },
12        'air_pressure': {
13            'dims': ['mid_levels', '*'],
14            'units': 'Pa',
15        },
16    }
17
18    diagnostic_properties = {}
19
20    tendency_properties = {
21        'air_temperature': {
22            'dims': ['mid_levels', '*'],
23            'units': 'degK s^-1',
24        }
25    }
26
27    def __init__(self, forcing_filename, **kwargs):
28        [...]
29        super(PrescribedHeating, self).__init__(**kwargs)
30
31    def array_call(self, state):
32        [...]

```

Readability

1. **Clear and Concise Language:** Use simple and straightforward language to describe food items, portion sizes, and nutritional information. Avoid jargon or technical terms unless the target audience is well-versed in nutrition.
2. **User-Friendly Interface:** Design an intuitive and user-friendly interface with clear navigation and well-organized content. Use familiar icons and symbols for common actions like adding food, viewing logs, or setting goals.

3. **Categorization and Filters:** Organize food items into categories (e.g., fruits, vegetables, proteins) and provide filters to help users quickly find the items they want to track. A search bar can also enhance usability.
4. **Visual Aids:** Incorporate images or icons to represent food items. These visuals can help users identify and select the right items more easily. For instance, a picture of an apple for an apple entry.
5. **Serving Size Options:** Provide different serving size options (e.g., grams, ounces, cups) and use measurement units that are familiar to the user. Make it easy for users to adjust portion sizes.
6. **Nutritional Information:** Present nutritional information, such as calories, macronutrients, and micronutrients, in a clear and organized format. Consider using tables or easy-to-read labels.
7. **Meal Tracking:** Allow users to input meals (breakfast, lunch, dinner, snacks) and display meal summaries. This helps users see their daily intake in a more meaningful context.
8. **Progress Tracking:** Display users' progress towards their goals in a visually appealing way. Charts and graphs can be useful for showing trends over time.
9. **Feedback and Suggestions:** Provide feedback on users' food choices and suggest healthier alternatives when appropriate. Use a friendly and non-judgmental tone in your suggestions.
10. **Mobile Responsiveness:** Ensure that the system is mobile-responsive, as many users may want to track their food on the go. Optimize the design for both web and mobile devices.
11. **Customization:** Allow users to personalize their experience by setting their goals, dietary preferences, and health parameters. Personalization can improve user engagement and satisfaction.
12. **Help and Support:** Include clear and accessible help sections, FAQs, and support options (e.g., chat support) to assist users with any questions or issues they may encounter.
13. **Accessibility:** Ensure that your system is accessible to all users, including those with disabilities. This includes providing alt text for images and ensuring compatibility with screen readers.
14. **Testing and Feedback:** Continuously gather user feedback and conduct usability testing to identify and address any readability or usability issues.

By following these principles and tips, you can create a food tracking system that is not only functional but also user-friendly and easy to read, ultimately enhancing the user experience and encouraging users to make healthier food choices.

Reusability

1. **Modular Design:** Divide your food tracking system into smaller, independent modules or components that serve specific functions. Each module should be designed to perform a specific task, such as data input, data processing, user authentication, or reporting. By

making these modules reusable, you can use them in different parts of the system or even in other applications.

2. **Libraries and Frameworks:** Leverage existing libraries, frameworks, or APIs that offer food-related functionality. For example, you can use third-party libraries for barcode scanning, nutrition data retrieval, or recipe management. This allows you to build on the work of others and save time and effort in development.
3. **RESTful APIs:** Implement a RESTful API for your food tracking system. This API can serve as a standardized way to interact with your system, allowing you to reuse it for different client applications (e.g., mobile apps, web apps, or IoT devices). By adhering to REST principles, you make it easier for developers to understand and use the API.
4. **Database Schema Design:** Create a well-structured and normalized database schema that separates data logically. This way, you can reuse the database design for other applications that require similar data storage and retrieval, such as a recipe management system or a calorie tracking app.
5. **User Interface Components:** Design reusable user interface components or widgets that can be used across different screens or platforms. This can include common elements like input forms, buttons, and data visualization components.
6. **Configurability:** Build configurability into your system to accommodate various use cases. Allow users or developers to customize aspects of the system, such as units of measurement, dietary preferences, or goal tracking. This flexibility can make your system more adaptable and reusable for a broader audience.
7. **Version Control:** Implement version control and documentation practices to keep track of changes, updates, and enhancements in your system. This ensures that changes are made in a way that maintains backward compatibility, making it easier to reuse existing components.
8. **Open Source:** Consider making parts of your food tracking system open source. This can encourage collaboration and contributions from the developer community, leading to improvements and additional features that benefit everyone.
9. **Integration with External Systems:** Design your system to integrate with external systems and services, such as fitness trackers, recipe databases, or restaurant review platforms. By creating well-defined integration points, you can easily connect your system to new data sources or services, expanding its functionality.
10. **Documentation:** Provide clear and comprehensive documentation for your system, APIs, and components. Well-documented code and APIs make it easier for other developers to understand and reuse your work.