SD²: Software Design Document

Github Link: https://github.com/WillyLeung1/Daily-Meal-Planner.git

Document is titled: “Team6_D2”

Installation directions are on the README file on Github.

1) Project Overview

Problem statement:

The Daily Meal Planner software addresses the challenge of creating balanced, time-efficient meal plans tailored to users' dietary preferences and nutritional needs. Many individuals face difficulty in organizing meals that fit within their daily schedule, align with caloric requirements, and consider dietary restrictions. The Daily Meal Planner simplifies this process by offering recipe recommendations based on available cooking time, calculating calorie counts for each meal plan, and enabling users to switch food items while maintaining dietary balance. This application ultimately helps users adhere to healthier eating habits without the stress of extensive planning.

Stakeholders and Their Stakes

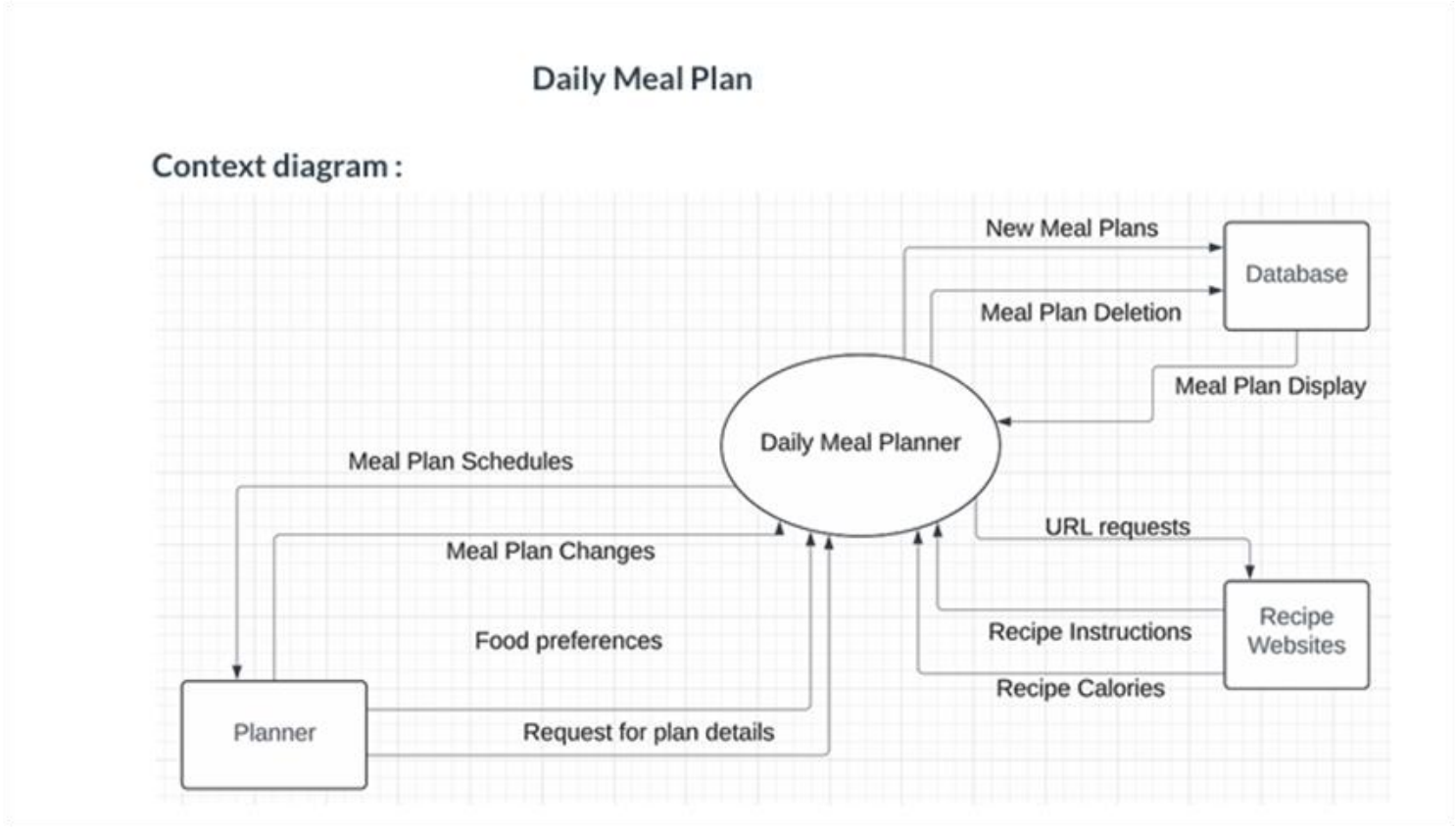
- Primary Stakeholders:
  - Planner (User): The main user who interacts with the system to create and customize meal plans. They rely on accurate recipe recommendations, nutritional information, and flexible meal planning options to fit their daily routines.
  - Nutritionists: Interested in providing accurate dietary information to clients, they support users' calorie management and nutritional tracking efforts.
  - Health-Conscious Users: Individuals focused on maintaining a healthy diet by carefully planning meals and monitoring calories to support their wellness goals.
- Secondary Stakeholders:
  - Development Team: Responsible for the technical implementation, ensuring the system is user-friendly and accurately retrieves and presents nutritional data.
  - API Providers (e.g., Edamam Recipe Search API): Provides access to a diverse range of recipes and nutritional data, enabling the application to offer robust recipe suggestions and caloric tracking.

Solution Overview

The Daily Meal Planner application provides a structured approach to meal planning, enabling users to select meals based on cooking time, view nutritional and caloric data, and customize plans to fit their preferences. By integrating the Edamam Recipe SearchAPI, the application accesses abroad selection of recipes with detailed nutritional information, allowing for personalized, time-based recipe suggestions.

Additional features include viewing past meal plans, switching food items, and tracking daily caloric intake, making the planner a comprehensive tool for managing a healthy diet.

General Models:



UseCaseDiagram:

User Stories

1. As a Planner, I want to view recipe recommendations based on my available cooking time, so that I can find suitable meals quickly.

2. As a Health-Conscious User, I want to view calorie counts for each item in my meal plan, so I can manage my intake effectively.
3. As a Planner, I want to create a custom meal plan by specifying the number of days, so I can have a tailored meal plan that fits my schedule.
4. As a Planner, I want to replace a food item in my meal plan, so I can adapt my meals without recreating the entire plan.
5. As a Planner, I want to view past meal plans, so I can review my previous meal choices and adjust future plans accordingly.

Product Backlog

Modify User Input Interface for Time Estimation

- Description: Design and implement a front-end interface to capture the user's cooking time estimation.
- Subtasks:
    - Prototype Design: Create an input form prototype for user input.
    - Frontend Implementation: Develop the front-end form for input.
    - Backend Integration: Pass user-input time to the backend database.
- Team Member: Fully implemented by Zhongxing Jordan Qin
- Effort:7-8/10
- Progress: Complete

Implement Custom Meal Plan Feature

- Description: Add a "Custom Plan" option that lets users select specific days for personalized meal planning.
- Subtasks:
    - Logic and Backend Integration: Handle custom plan logic and integrate with backend.
    - Frontend Improvement: Enhance the frontend to support custom plan selection.
- Team Members: Assigned to Sarayu Pacca, fully implemented by Willy
- Effort:5-7/10
- Progress: Complete

Implement Bug Fix for Number of Meals Display

- Description: Debug and fix the display to ensure the correct number of meals is reflected as chosen in the meal plan.
- Team Member: Willy Leung
- Effort:6/10
- Progress: Scrapped, issue is believed to reside in the Edamam DB, making it extremely difficult to debug

Implement Food Substitution Feature

- Description: Develop an interface allowing users to switch out food items in a generated meal plan with alternatives while updating nutritional information.
- Subtasks:
    - User Interface for Food Substitution: Design the UI for selecting and replacing food items.
    - Prompt for Substitution: Prompt users to specify which food they wish to replace.
    - No limits to Meal Replacement.
- Team Members: Assigned to Mrinal Raj, fully implemented by Zhongxing Jordan Qin
- Effort:6-8/10
- Progress: Complete

Show Calories for Each Food in Meal Plan

- Description: Implement a feature to display calorie information for each item in the meal plan.
- Subtasks:
    - Context for Calorie Counts: Provide calorie counts next to each food item.
- Team Members: Assigned to Maheswari, fully implemented by Willy
- Effort:6/10
- Progress: Complete

Save and Manage Meal Plans

- Description: Create a feature that automatically saves meal plans and enables users to access them in a history tab, with options to delete plans.
- Subtasks:
    - Backend and Database Setup: Store meal plan data in the database.
    - Delete Functionality: Add options to delete past meal plans and update the database.
    - Sort Function: Sort meal plans based by date
- Team Members: Fully implemented by Willy
- Effort:4-8/10
- Progress: Complete

Create a log-in page to connect to MongoDB

- Description: Create a log-in page where users can input their MongoDB connection string so they can connect to their own clusters to store data.
- Subtasks:
    - Front-end work: Create a page to take user input.
    - Back-end: Take the user-input string, confirm its validity, and use it to connect to the database.
- Team Members: Assigned to Mrinal Raj
- Effort:10/10
- Progress: Incomplete, unable to be completed in the timeframe since the group critiques.

Main Use Cases (Technical Details)

1. Recommend Recipes Based on Available Cooking Time

· Description: The system uses the Edamam Recipe Search API to retrieve recipes based on the user's specified cooking time, filtering results to match their time constraints.

· Pre-condition: The Edamam API is accessible and integrated with the application.

- Trigger: User accesses recipe recommendations and inputs their available time.
- Post-condition: User views API-recommended recipes that fit with in the specified cooking time.

2. View Caloric Information for Meal Plans

   - Description: For each food item in a meal plan, the system displays calories using data from the Edamam API, enabling accurate tracking of daily calorie intake.
   - Pre-condition: The Edamam API is functional, and each meal plan item is associated with a recipe ID or calorie data from the API.
   - Trigger: User selects to view calories for each food item in the meal plan.
   - Post-condition: The application displays caloric information from the Edamam API alongside each item in the meal plan.

3. Create Custom Meal Plan

   - Description: User specifies the meal plan duration and preferences; system generates a custom meal plan based on these inputs.
   - Pre-condition: The "Custom Plan" option is available to the user.
   - Trigger: User selects "Custom Plan" and specifies preferences.
   - Post-condition: System displays a customized meal plan according to the specified preferences.

4. Switch Out Food in Meal Plan

   - Description: User replaces a food item in a meal plan with an alternative, updating the calorie count and nutritional values using Edamam data.
   - Pre-condition: User has a generated meal plan with food items.
   - Trigger: User selects "Switch Food" next to a food item.
   - Post-condition: Meal plan updates to reflect the swapped item and revised nutritional data.

5. View Past Meal Plans

   - Description: User views previously created meal plans stored in the system’s cloud database.

- Pre-condition: Meal plans are stored in the cloud database.
- Trigger: User navigates to the history tab.
- Post-condition: User can access and view details of past meal plans.

Use Cases for Daily Meal Planner

1. Recommend Recipes Based on Available Cooking Time

- Primary Actor: Planner (user)
- Description: Allows the user to input available cooking time, and the system suggests recipes that fit within that time frame.
- Trigger: User enters cooking time on the recipe recommendation page.
- Outcome: System displays recipes that can be prepared with in the specified time.
- Alternate Course: If no matching recipes are found, the system in forms the user.

2. Calories for Each Food Item in Meal Plan

- Primary Actor: Planner
- Description: The user can view calorie counts for each item in their meal plan, helping them manage calorie intake.
- Trigger: User selects "Show Calories" on the meal plan.
- Outcome: The system displays calorie information for each food item, including the total calorie count for the meal plan.
- Alternate Course: If data retrieval fails, the system shows an error.

3. Custom Meal Plan Option

- Primary Actor: Planner
- Description: Enables the user to create a custom meal plan by selecting a specific number of days.
- Trigger: User selects the "Custom Plan" option.
- Outcome: System generates a meal plan based on the user’s preferences.
- Alternate Course: If the user input is invalid, the system prompts for corrections.

4. Ensure Correct Number of Meals Displayed per Day

- Primary Actor: Planner
- Description: Ensures the meal plan reflects the exact number of meals per day as specified by the user.
- Trigger: User creates a meal plan.
- Outcome: System displays the correct number of meals per day.
- Alternate Course: If the system displays an incorrect count, an error message prompts the user to recreate the plan.

5. Switch Out Food in Meal Plan

- Primary Actor: Planner
- Description: Allows the user to substitute a food item in a meal plan with an alternative, updating nutritional values.
- Trigger: User selects the "Switch Food" option.
- Outcome: Meal plan is updated with the new food, and nutrition info is recalculated.
- Alternate Course: If no alternative is available, the system notifies the user.

6. View Past Meal Plans in a History Tab

- Primary Actor: Planner
- Description: Saves and all ows access to previously generated meal plans.
- Trigger: User navigates to the history tab.
- Outcome: User can view and select past meal plans.
- Alternate Course: If there's an error in fetching data, an error message is displayed

2) Architectural Overview

1. **General Design**:

   This system's architecture includes a few main parts, each designed to make meal planning easy for the user:

   - A place for users to input cooking time
   - An option for custom meal planning
   - A feature to swap out foods in meal plans
   - Away to show calorie counts and save meal plans

   Each feature was built to be simple and separate, making it easier to update parts of the system without affecting others.

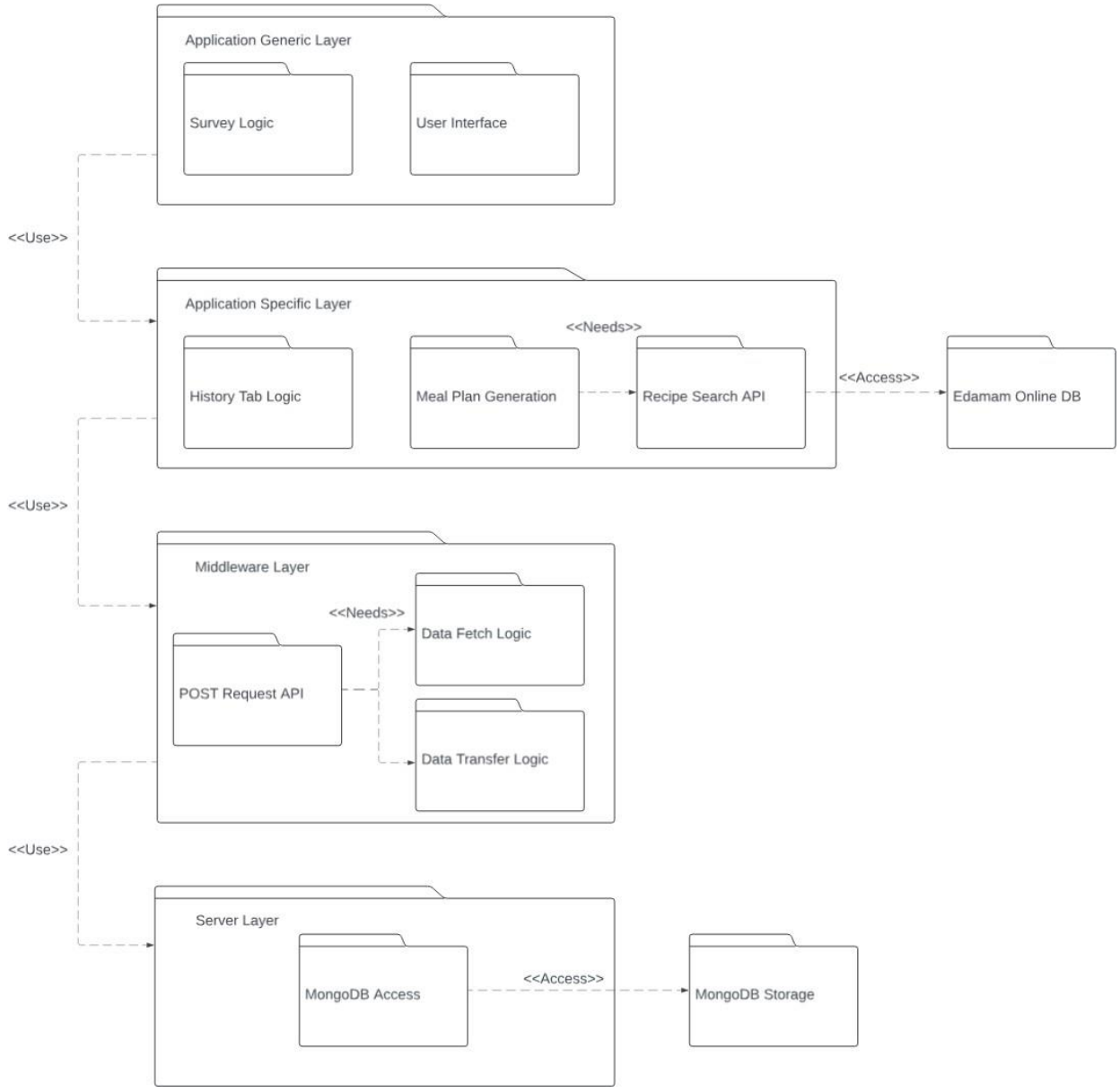
2. **Alternative Designs Considered**:

   - Different ways to create some of these features were considered. For example, we looked at other options for the meal substitution and calorie display features.
   - Some ideas were set aside because they were too complicated, didn't fit well with the other parts, or would have taken too long to build.

3. **Why This Design Was Chosen**:
   - We chose this design because it's easy to manage, works smoothly for users, and allows us to add updates more easily in the future.
   - Keeping each feature separate (like meal substitutions and saving meal plans) make sit easier to fix or improve each part individually.

Subsystem Architecture

UML Package Diagram:



The above UML Class diagram contains four layers: application-specific, application-generic, middleware, and server. We categorized the packages into one of each layer because most of these packages work together in each layer, and rarely with packages from other layers (except for both application layers, in this case). A layered architecture enables a separation of roles for many components, and this design approach allows team members to code independently without their code affecting the rest of the system's components, making debugging, implementation, and integration easier. We also created subsystems to group packages together, and this design approach enables us to take into

account the involved packages we need to think about when we make changes, since the subsystem packages are dependent or used by each other.

In the **Application Generic layer**, there are Survey Logic and User Interface packages. The User Interface package controls the visual layout of the application, and the Survey Logic is responsible for how the survey questions appear and how user inputs are taken. The two packages are related because the survey needs a UI for users to provide inputs, hence they are in the same subsection.

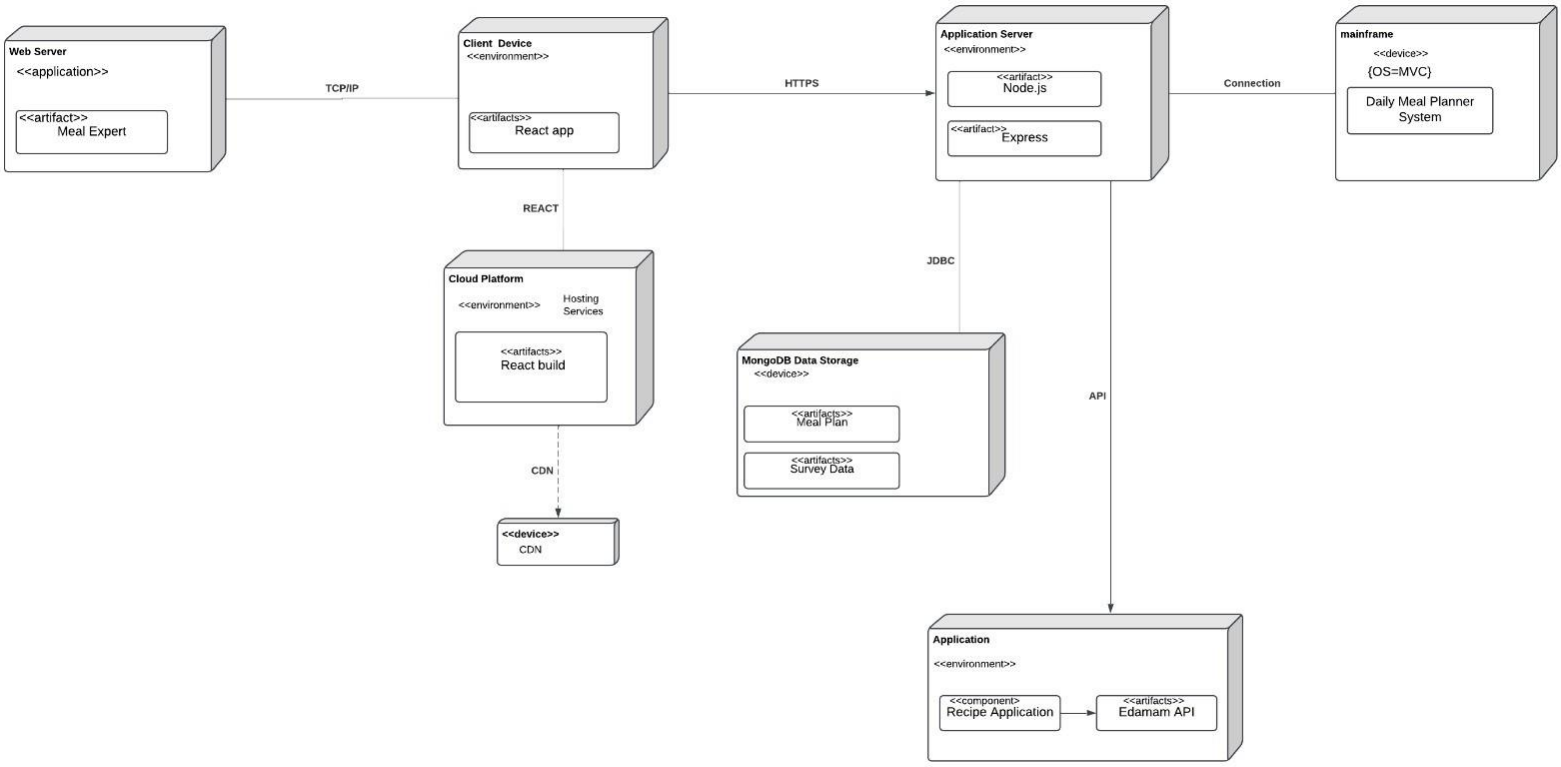
In the **Application Specific layer**, there are the History Tab Logic, Meal Plan Generation, Recipe Search API, and the Edamam Online DB. The History Tab Logic is responsible for displaying the past meal plans. The Meal Plan Generation package is responsible for taking the user inputs to create a meal plan, which is why it needs the Recipe SearchAPI.TheRecipeSearchAPIqueriestheEdamamOnlineDatabaseforthereciperecipeURLs to create the meal plan with. Because the database is not a part of the system, it is outside the subsystem.

The **Middleware layer** contains the POST Request API, which contains the logic to fetch data from the front-end (Data Fetch Logic) and the logic to send data to the back-end (Data Transfer Logic) The API will process POST requests from the front-end to retrieve both the survey and meal plan data to be sent to the back-end.

The **Server layer** contains the MongoDB Access package, which is responsible for the logic to access the MongoDB Atlas database. Here in the back-end, the data from the survey and meal plan will be formatted into a JSON format and sent to MongoDB to be stored. Since MongoDB is outside the system, it is not a part of the subsystem.

Deployment Architecture

Deployment Diagram:



The UML deployment diagram for the Daily Meal Planner project illustrates the architecture and interactions between various components:

- Web Server: Hosts the "Meal Expert " application, which communicates with client devices via HTTPS.
- Client Device: Runs a React app, accessing the webserver and application server over HTTPS.
- Application Server: Utilizes Node.js with Express to handle requests. It connects to the database server using JDBC and interacts with the mainframe for meal planning.
- Main frame: Hosts the "Daily Meal Planner System" and connects to the application server.
- Database Server: Contains the "Recipe Database " and communicates with the application server via JDBC.

- Cloud Platform: Provides hosting services for the React build, distributing it through a Content Delivery Network (CDN).
- Edamam API Server: An external application that connects to the database server ia API for additional data retrieval.
- MongoDB Data Storage: An external storage database to connect to the backend server via JDBC to transfer survey and meal plan data.

This architecture ensures secure communication, efficient data handling, and integration with external services for meal planning functionality.

Persistent Data Storage

The approach we use to store data is to store JSON files on MongoDB Atlas. We take both the survey data and meal plan data, and combine them together into a hierarchical JSON file before sending it to MongoDB for storage. The hierarchy is shown in the following form:

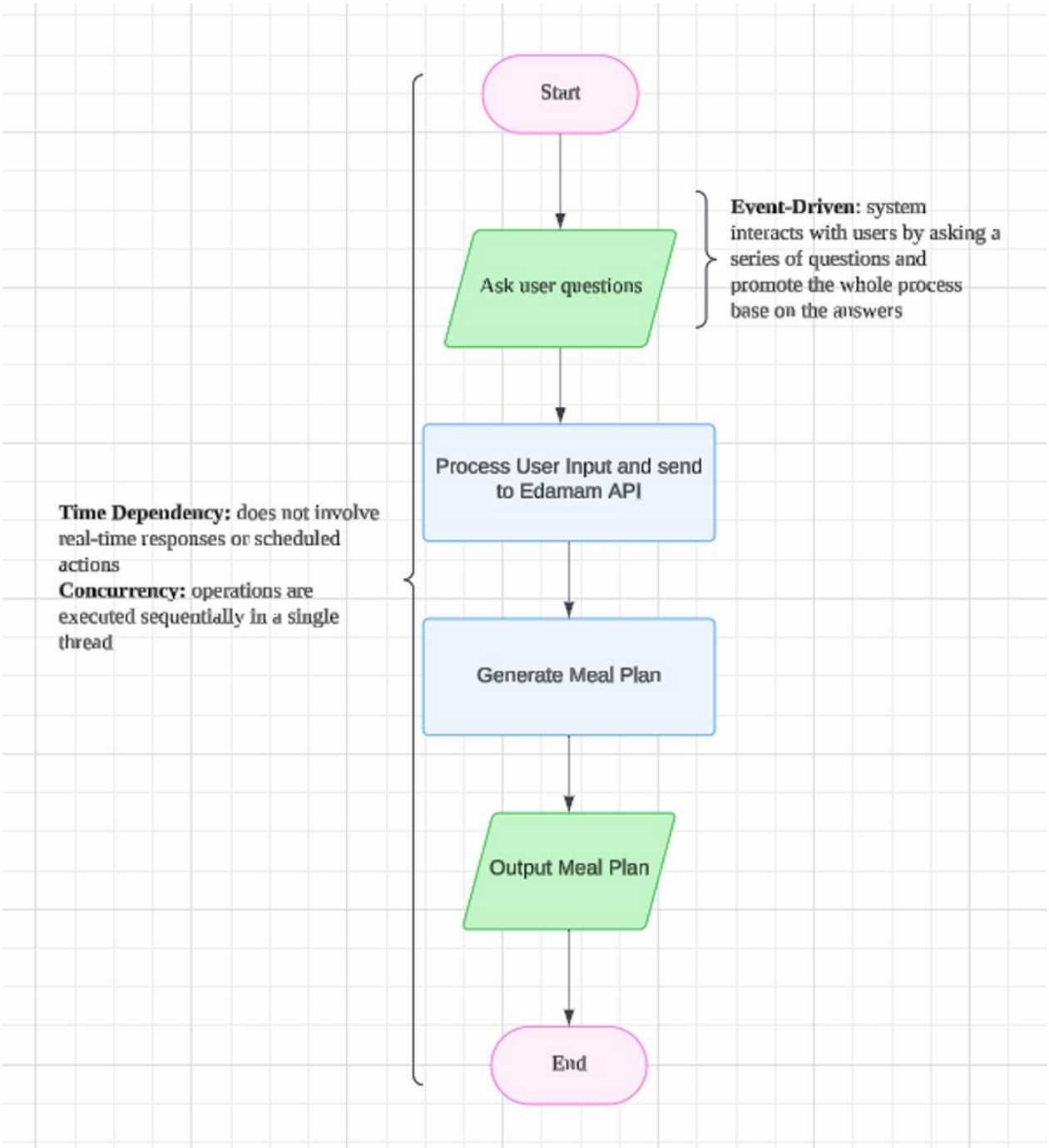
```
ObjectID: String

surveyData: Object
       daysInPlan: int (M)
       Calories: Object
              min: int
              max: int
       Diet: String
       cookingTime: int
       meals: Array (N)
              0: String (i.e. "Breakfast")
              1: String (i.e. "Lunch")
              2: String
              …
              N: String

mealPlanData: Object
       Breakfast: Array(M)
       Lunch: Array (M)
       Dinner: Array (M)
```

The meal Plan Data object contains arrays of the size of the days In Plan integer, which further contains objects for each individual meal for a particular day. Expanding upon the meal object will yield all details of the meal, including tags, ingredients, caution warnings, and

instructions on how to cook. These labels are taken directly from the Edamam online database from the recipe search API. Below is a screenshot of a sample meal object:

GlobalControlFlow

ControlFlowDiagram:



Procedural or Event-driven: This system is event-driven. The meal planner system interacts with users by asking a series of questions about their dietary preferences, target calories,

and other goals. Users can respond to these questions in a sequential manner, and the system generates a personalized meal plan based on their inputs. Each user's interactions may vary, and the system responds dynamically to user inputs.

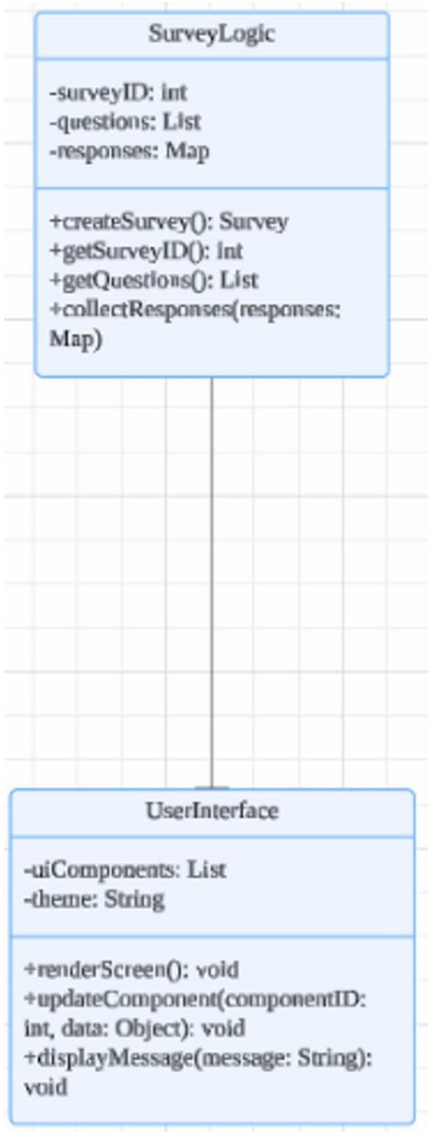
Time Dependency: This system does not have time dependencies. The process of generating a meal plan is solely based on user input and does not involve real-time responses or scheduled actions.

Concurrency: This system does not require concurrency; all operations are executed sequentially in a single thread. There is no need for multi-threading or synchronization mechanisms in this system.

3) Detailed System Design

We will base our static views and dynamic views off of the package diagram in 2.1. The critical components will be based on the subsections in each layer of the diagram. Our reason for this is that those subsections contain the necessary code and logic to perform their tasks within the system, so they can be considered standalone components of our architecture.

Application-Generic Component:

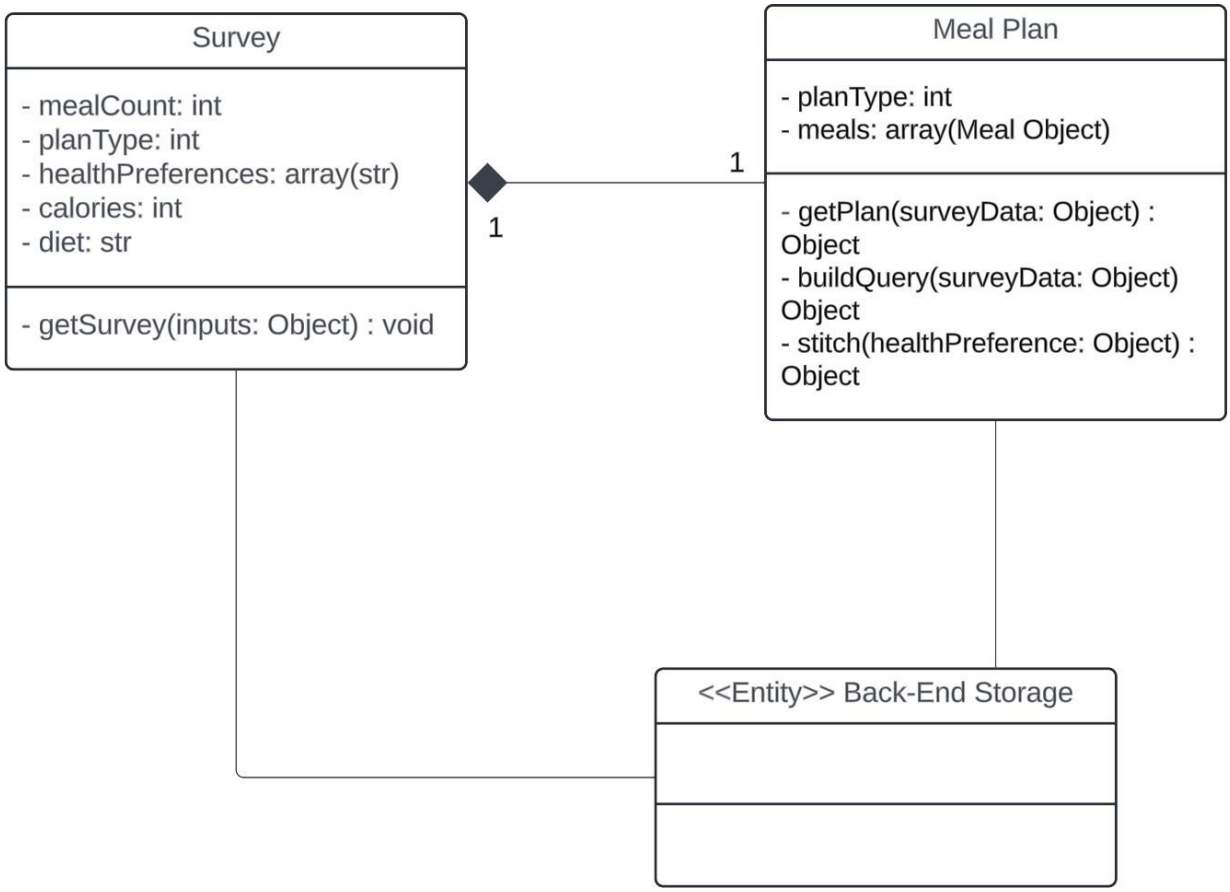


Survey Logic manages the creation of surveys and the collection of user responses. It serves as the logic layer for handling surveys and user inputs. UserInterface handles the user interface and display logic. It is responsible for presenting data to the user and updating the interface based on user interactions.

Application-Specific Component:

RecipeSearchAPI interacts with an external recipe database (such as the Edamam API), allowing for recipe searches and retrieval of detailed information. Meal Plan Generation is responsible for generating personalized meal plans based on the user's survey data. History Tab Logic manages the history of meal plans, allowing users to view, add, or delete historical records.

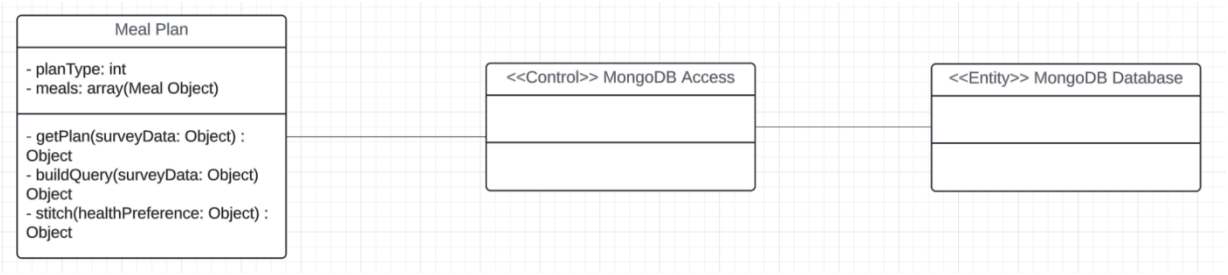
Middleware Component:



The above diagram is a breakdown of the MealPlanGeneration component from the previous diagram. The POST Request Send logic is embedded in both tables shown above. These files are responsible for obtaining the data for both survey and meal plan, so we also added logic to the field to send data to the back-end storage.

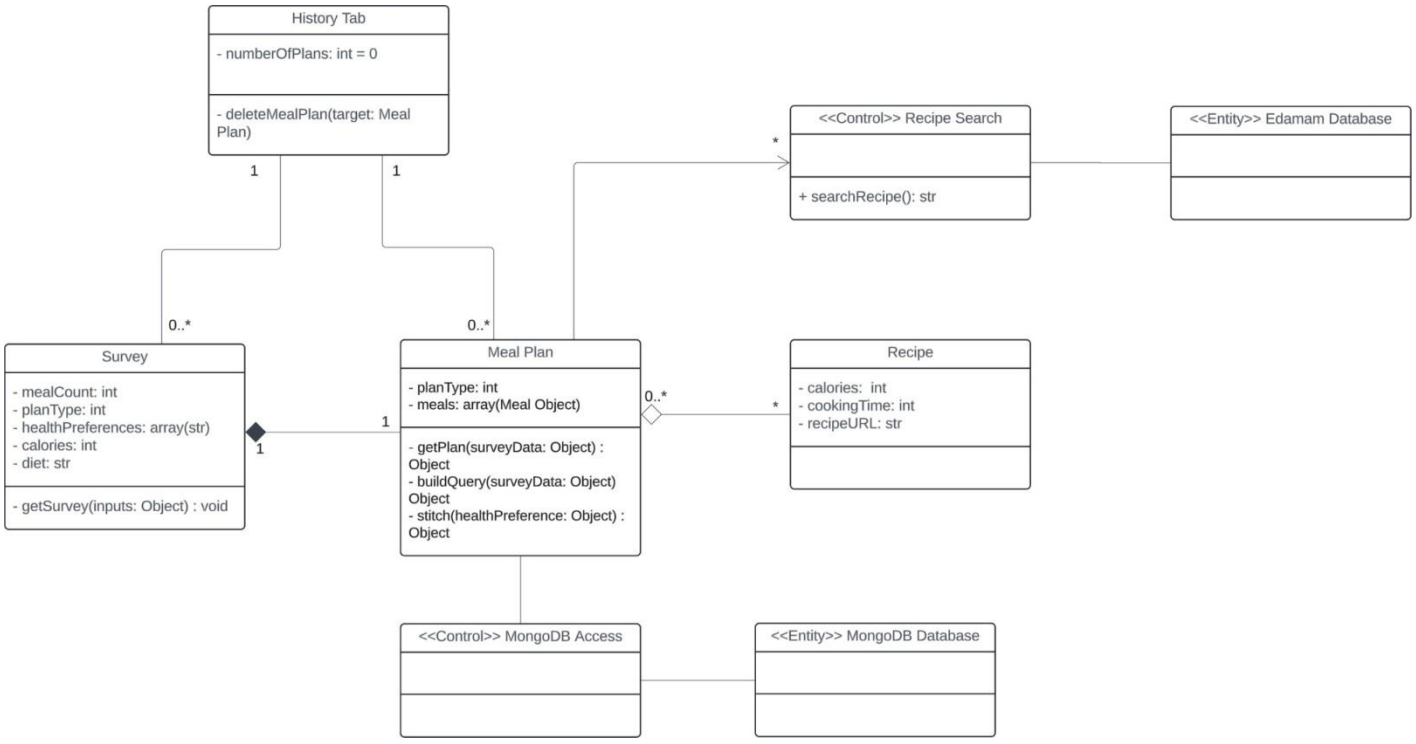
Survey Data contains data about the user inputs from the survey, and contains a method that will retrieve the user inputs and send it to the back end with POST requests. Meal Plan Data contains the data about the meal plan itself, and uses getPlan() to generate a meal plan by obtaining survey data(from getSurvey()).The function build Query() is used with in get Plan to fetch recipe data from Edamam, and stitch() formats the health Preference inputs.

Server Component:



The back-end server simply takes the data and connects to MongoDB for data transfer. No classes exist for the MongoDB Access and databases, so it is simply an association with an external entity and the back-end logic.

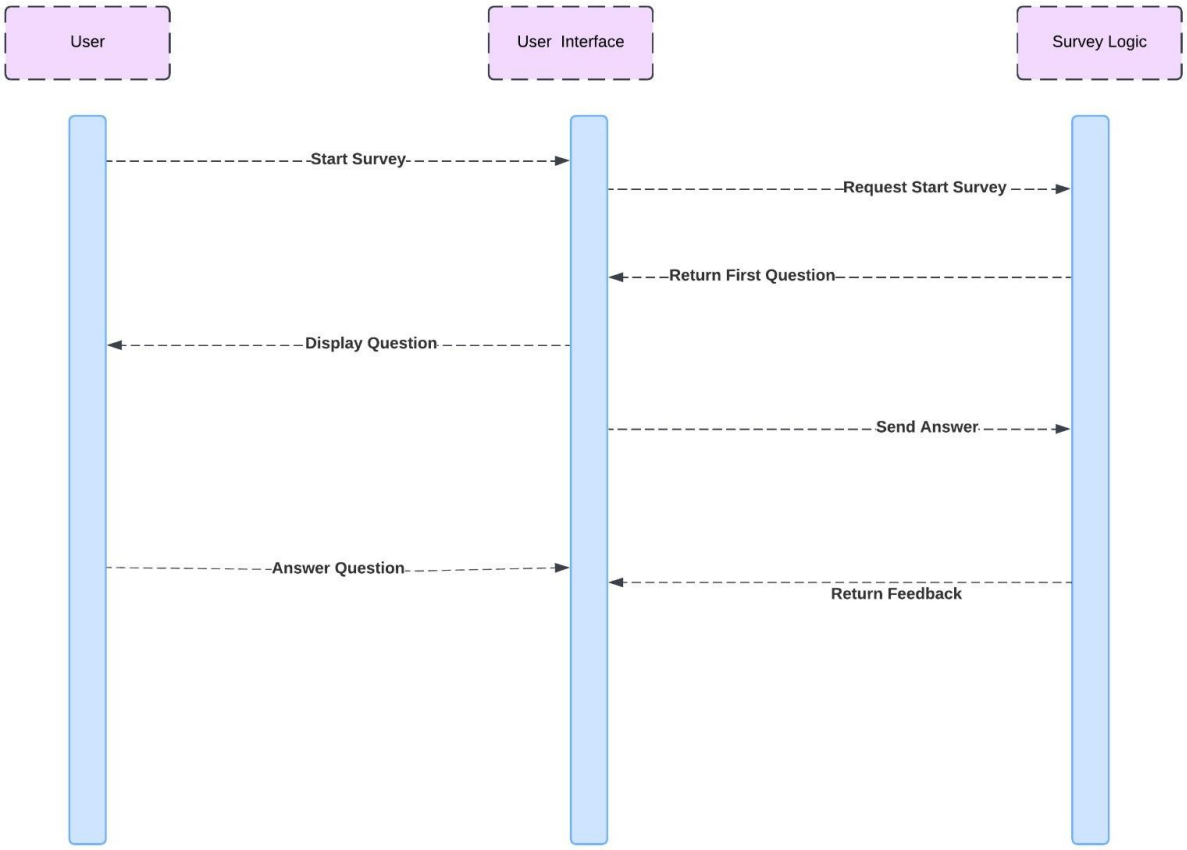
Combined Components:



The above diagram combines front-end and back-end components together, showing the meal generation process from the Application-Specific layer, and the connections to external entities. We decomposed them into smaller tables to show their specific workings in terms of each layer, especially since some tables can be further decomposed in to smaller modules. This diagram would give an overview of the previous diagrams.

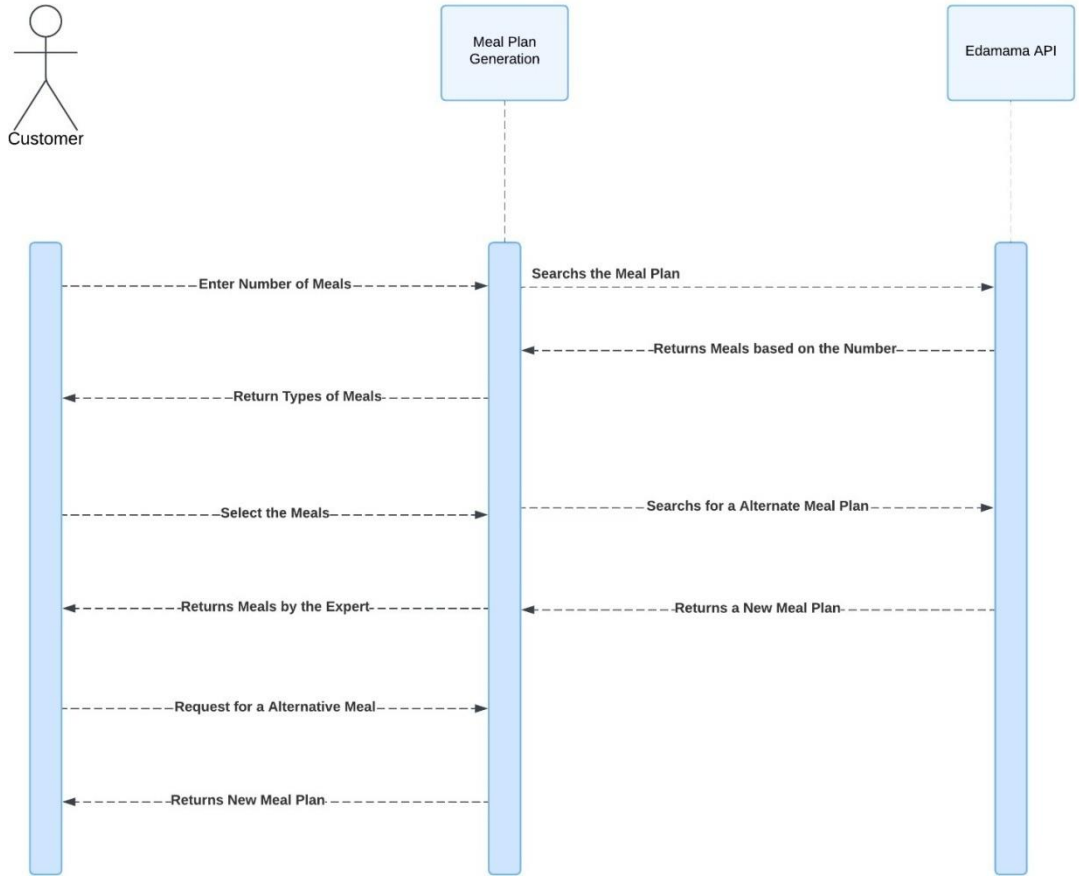
3.2 Dynamic view

Application-Generic Component:



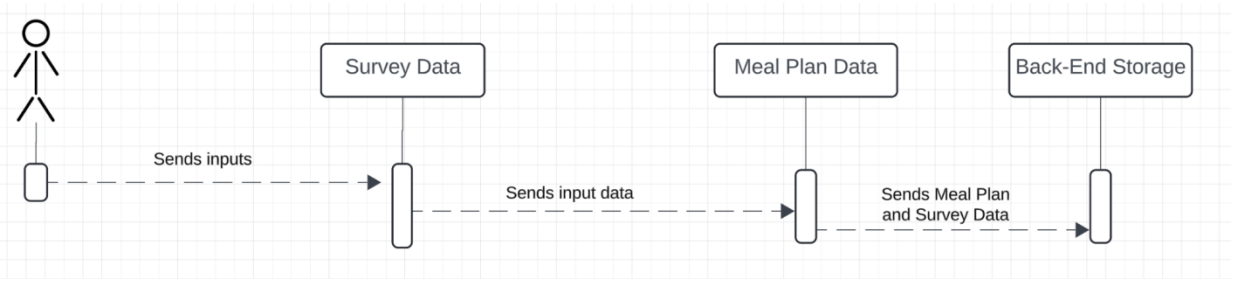
The above diagram shows how the survey handles presenting the question and taking responses. The User will initiate the survey, and then input responses; this is the only event-driven part of the system.

Application-Specific Component



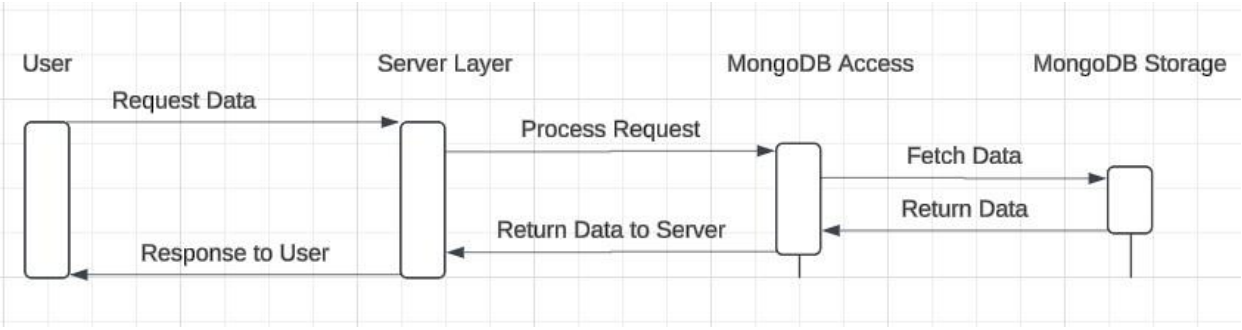
The above diagram shows how the system generates a meal plan. After the user inputs the responses, the system will search the Edamam Database and return the appropriate meals for the meal plan. As a feature, the user can alternatively switch out meals, and this diagram accounts for user actions in replacing meals.

Middleware Component:



The diagram is a very short and linear flow from the user to the back- end storage. The purpose of this diagram is to show how the POST request API handles the dataflow, and the flow is mainly in one direction.

ServerComponent:



The above diagram shows how the user's inputs and meal plan end upon MongoDB, and how the user can view the data again from MongoDB in a history tab.

Test Plans:

White-Box Testing**:**

- We successfully conducted white-box testing during each sprint. Team members utilized their assigned use cases to validate the functionality of the tasks they implemented. This ensured that each feature and function operated as intended throughout the development process.
- Any identified issues were resolved promptly, maintaining the integrity of the codebase and functionality after every sprint.

Black-Box Testing**:**

- Towards the end of the semester, we transitioned to black-box testing. New use cases

were created specifically for testing purposes, focusing on verifying the application's functionality from an end-user perspective.

- We successfully recruited individuals outside our development team to participate as testers. Their feedback provided valuable insights into the usability and overall user experience of our application.
- This process allowed us to identify and resolve usability issues that might not have been apparent during internal testing.

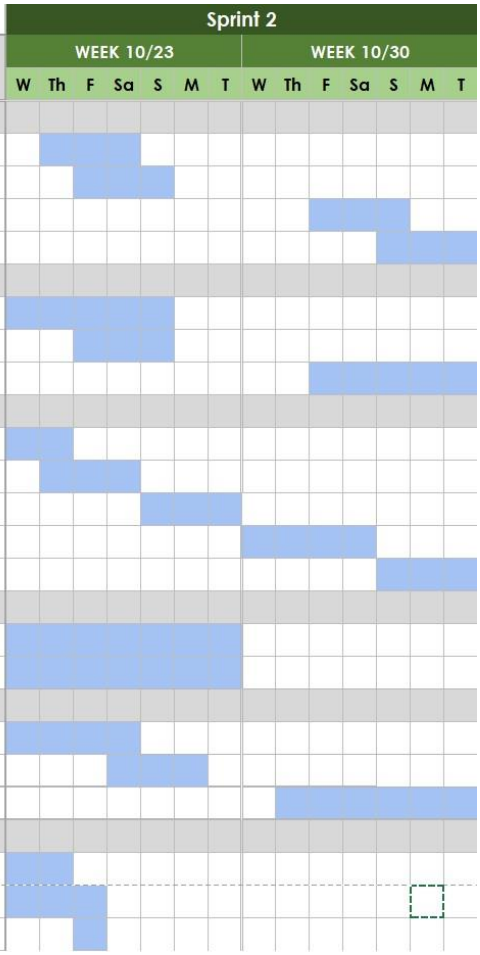
Sprint Reviews:

Sprint1:
Original Schedule for Sprint1:

							Sprint 1																						
TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	EXPECTED # OF DAYS	% OF TASK COMPLETE	WEEK 10/2							WEEK 10/9							WEEK 10/16								
							W	Th	F	Sa	S	M	T	W	Th	F	Sa	S	M	T	W	Th	F	Sa	S	M	T		
1	Modify User Input Interface for Cooking Time Estimation																												
1.1	Understand the code baseline structure	Jordan	10/04/24	10/06/24	3	100%																							
1.2	Design the input form prototype for user input	Jordan	10/07/24	10/09/24	3	100%																							
1.3	Build the frontend input form	Jordan	10/10/24	10/12/24	3	100%																							
1.4	passe the user' input time to the back-end database	Jordan	10/13/24	10/15/24	3	0%																							
1.5	Test the input feature to ensure it works as intended	Jordan	10/16/24	10/16/24	1	75%																							
2	Implement bug fixes to display # of plans																												
2.1	Review and locate affected files	Willy	10/05/24	10/07/24	3	100%																							
2.2	Debug and fix files	Willy	10/07/24	10/10/24	4	100%																							
3	Implement a back-end database to store meal plans																												
3.1	Set up and configure a MondoGD Atlas Cluster	Willy	10/09/24	10/10/24	2	100%																							
3.2	Write a JS file to connect to the DB using Node.js	Willy	10/10/24	10/13/24	4	100%																							
3.3	Store generated meal plans	Willy	10/13/24	10/15/24	3	50%																							
3.4	Transfer generated meal plans to DB	Willy	10/15/24	10/18/24	4	50%																							
3.5	Test the code	Willy	10/19/24	10/19/24	1	0%																							
4	Enhance plan type selection with expanded user options																												
4.1	Understand the code.	Sarayu	10/05/24	10/07/24	3	100%																							
4.2	Create custom plan option in UI	Sarayu	10/08/24	10/10/24	3	50%																							
4.3	Design user input for custom days	Sarayu	10/12/24	10/14/24	2	25%																							
4.4	Basic frontend logic for custom plan	Sarayu	10/15/24	10/17/24	3	0%																							
5	Implement method to switch out food in plan																												
5.1	Recode the program to fetch another food item	Mrinal	10/12/24	10/18/24	7	50%																							
5.2	Add switch out buttons to each food item in plan	Mrinal	10/13/24	10/15/24	3	0%																							
6	Implement a method for a user to save meal plans																												
6.1	Create a "history" tab in the main page	Willy	10/19/24	10/19/24	1	0%																							
6.2	Display previous meal plans from the DB in this history pag	Willy	10/20/24	10/22/24	3	0%																							
7	Implement a way to show calories for each food in the meal plan																												
7.1	Research and select a calorie data source	Sarayu, Maheswa	10/10/24	10/12/24	3	40%																							
7.2	Display the calories on the meal page	Maheswari	10/13/24	10/16/24	4	0%																							

Updated Schedule for Sprint2 (Tasks moved from Sprint1 is denoted with an S1):

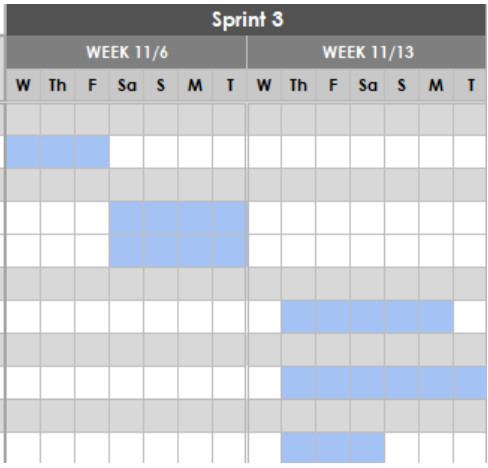
TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION IN DAYS	% OF TASK COMPLETE
1	Implement a method for a user to save meal plans (CONT.)					
S1.6.1	Create a "history" tab in the main page	Willy	10/24/24	10/26/24	3	0%
S1.6.2	Display previous meal plans from the DB in this history pag	Willy	10/25/24	10/27/24	3	0%
1.1	Add delete buttons to delete past meal plans	Willy, Jordan	11/01/24	11/03/24	3	0%
1.2	Update the data in DB to reflect deleted data	Willy, Jordan	11/03/24	11/05/30	3	0%
2	Implement method to switch out food in plan (CONT.)					
S1.5.1	Recode the program to fetch another food item	Mrinal	10/23/24	10/27/24	5	50%
S1.5.2	Add switch out buttons to each food item in plan	Mrinal	10/25/24	10/27/24	2	0%
2.1	Prompt user with questions to specify which food they wo	Mrinal	11/01/24	11/05/24	5	0%
3	Enhance plan type selection with expanded user options					
S1.4.2	Create custom plan option in UI	Sarayu	10/23/24	10/24/24	2	50%
S1.4.3	Design user input for custom days	Sarayu	10/24/24	10/26/24	3	25%
S1.4.4	Basic frontend logic for custom plan	Sarayu	10/26/24	10/28/24	3	0%
3.1	Handle custom plan logic and Backend integration	Sarayu	10/30/24	11/02/24	4	0%
3.2	Validation and Testing	Sarayu	11/3/24	11/05/24	3	0%
4	Prepare for Demo on 10/30					
4.1	Review code (refactor, simplify, debug)	All teams	10/23/24	10/29/24	7	0%
4.2	Integrate code into Github	Willy, All teams	10/23/24	10/29/24	7	0%
5	Implement a way to show calories for each food in the meal plan					
S1.7.1	Research and select a calorie data source	Sarayu, Maheswa	10/23/24	10/26/24	4	40%
S1.7.2	Display the calories on the meal page	Maheswari	10/26/24	10/28/24	3	0%
5.1	Provide context for calorie counts	Sarayu, Maheswa	10/31/24	11/05/24	6	0%
S1.3	Implement a back-end database to store meal plans					
S1.3.3	Store generated meal plans	Willy	10/23/24	10/24/24	2	50%
S1.3.4	Transfer generated meal plans to DB	Willy	10/23/24	10/25/24	3	50%
S1.3.5	Test the code	Willy	10/25/24	10/25/24	1	0%



Updated backlog is present in section 1.

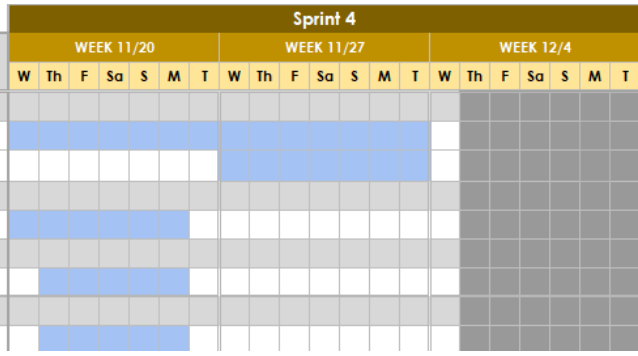
Updated Schedule for Sprint3

TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION IN DAYS	% OF TASK COMPLETE
1	Implement a method for a user to save meal plans (CONT.)					
1.1	Add a confirm window for deletion (i.e. "Are you sure?")	Willy, Jordan	11/06/24	11/08/24	3	100%
2	Prepare for Demo on 11/13					
2.1	Review code (refactor, simplify, debug)	All teams	11/09/24	11/12/24	4	100%
2.2	Integrate code into Github	Willy, All teams	11/09/24	11/12/24	4	100%
3	Implement method to switch out food in plan (CONT.)					
3.1	Update DB to reflect food switched out in current meal plan	Mrinal, Willy	11/14/24	11/18/24	5	100%
4	Enhance plan type selection with expanded user options					
4.1	Improve the frontend for plan type selection.	Sarayu	11/15/24	11/20/24	6	100%
5	Implement a way to show calories for each food in the meal plan					
5.1	Improve the frontend of the page.	Maheswari	11/15/24	11/17/24	3	100%



Updated Schedule for Sprint 4:

TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION IN DAYS	% OF TASK COMPLETE
1	Prepare for presentation on 12/4					
1.1	Review code (refactor, simplify, debug)	All teams	11/20/24	12/03/24	14	100%
1.2	Integrate code into Github	Willy, All teams	11/27/24	12/03/24	6	100%
2	Implement a method for a user to save meal plans (CONT.)					
2.1	Implement way to view two meal plans side-by-side	Willy, Jordan	11/20/24	11/25/24	5	100%
4	Enhance plan type selection with expanded user options					
4.1	Review code	Sarayu, All teams	11/21/24	11/25/24	5	100%
5	Implement a way to show calories for each food in the meal plan					
5.1	Review code	Maheswari, All tea	11/21/24	11/25/24	5	100%



Submission:

Submit your design documents inside your team Github repo. Moreover, **submit a PDF document in Canvas along with the link to repo; we will not grade any other document type**. Name your electronic submission as follows: **Team<number>_D2.pdf.** Submit your team assignment via Canvas. Only one team member needs to submit the document.

Tips:

Provide supplement al text to explain your diagrams through captions for them! Make sure that you have described, somewhere in the document, the responsibilities of the elements (e.g., components/modules/classes) in your architecture/class/sequence diagrams. You should describe your design decisions that led you to this design, including a discussion of any alternative designs you considered but discarded. Providing these kinds of descriptions helps in understanding your design (as such, they can have a positive impact on your grade!). *Where to Stop/When to Stop Drawing Interaction Diagrams*? Generating a sequence diagram for the most important user stories is typically a good way to start. If you find that you have a bunch of sequence diagrams that essentially repeat the same interaction, then you are not creating useful models, just redundant ones. In this case, generalize the interaction and provide supplemental text that explains how and where the generalized interaction can be applied to capture multiple user stories/use cases.
Remember, you should model only what is needed and useful as discussed in the class.

Apply Design Principles! You have learned about the importance of modules with high cohesion, a design with low coupling, and the benefits of relying on abstraction versus a concrete realization (e.g., application logic communicates with a hardware abstraction layer instead of directly with devices). Make sure that you apply these principles and clearly explain how your design achieves them.

Proofread your documents! Everyone on your team should proofread the document before it is submitted. It is important that we be able to understand your design to evaluate it, so you must take care in communicating your design. If you are not skilled in technical writing, plan in advance so that you can ask someone to proofread it for you. The university has a writing resource center that may be helpful to you.

What UML tools should I use to draw the diagrams? Any design tool may be used to draw your UML diagrams such as Draw.io and Lucidchart as we discussed in the class.

Be aware that while drawing programs may provide template tools for creating UML diagrams, those templates may not meet the diagramming guidelines we discussed in class. For instance, some versions of Microsoft Visio provided the wrong arrow for an “extends” relationship in the past for use-cases. You should check to make sure your diagram meets the standards discussed in class and make manual edits in the drawing program if necessary to adhere to those standards.