1. Team info

Provide a concise summary of the project team and project artifacts. Specifically:

- List each team member and their role in the project.

  Ayush Baruah, Backend Engineer / Tester

  Duncan Everson, Full-stack Engineer

  Benjamin Kono, Engineer in charge of UI design / Frontend Engineer

  Aiden McCoy, Engineer in charge of UI design / Frontend Engineer

  Daniel Molina, Backend Engineer

  Nicholas Shininger, Tester / Backend Engineer / Backend UI Engineer

- Link to each project relevant artifact such as your git repo (this can be empty for now).

  https://github.com/shiningn-osu/software-dev-ii-project

- List communication channels/tools and establish the rules for communication.

  Primary Asynchronous: Discord

  Primary Synchronous (outside of regular meetings): Teams

  Rules for Communication:

  To encourage positive/creative discussion, group members will refrain from interrupting others' ideas and from putting down any ideas during the discussion. It is expected that all group members speak respectfully of each other and allow others to voice their opinions without belittlement. To resolve disagreements the group members will seek to form a compromise between the two perspectives that led to the disagreement. In the case where compromise cannot be reached between the group members, then the group will seek third party input from a source outside of the group. The perspective that the third party selects will be the option that the group members use.

  For project management, we will use a GitHub Project Kanban board and roadmap. We will manage the tasks' status between ready, in progress, and done to reflect the current status of tasks. We will also assign tasks to individuals to reflect who is responsible for the task. Additionally, we will also put a size and time estimate on tasks to better divide up the workload between us. We will also note the start and end dates that these tasks involve so that we can better manage our time.

  If a team member gets confused about what to do, they should communicate that confusion in the Discord. If the confusion relates to something time sensitive, then *they can send an email as well. Synchronous Microsoft Teams meetings may also be scheduled if the group members deem such a meeting necessary to resolve the confusion.*

If a message sender sends a discord message, and they do not hear back within 12 hours (24 hours on the weekend), then they can send an email. This message may either reiterate the points of the discord message, or direct the recipient to the discord message. If there is no response to the email within 12 hours, then follow up text messages may be sent.

If a team member falls behind in their work, then they can post a message on Discord relaying that they are behind on their work. This message may request for either guidance or assistance from the other group member. In the event where the issue is time-sensitive or there is a large amount of work to catch up, an in-person meeting will be scheduled to dedicate time toward the unfinished work.

If a group member believes that they will fall behind due to upcoming exigent circumstances, then they are expected to relay that in the group Discord. From there, the group will redistribute the work such that the person who has the exigent circumstances has a workload that they can accomplish, and the other work has been divided among the group. The group's work redistribution allotments will be such that the redistribution is agreed as reasonable by all group members. For the next workload allotment, the individual that had exigent circumstances will take on an additional workload to make up for the previous reallotment such that the new allotment is agreed as reasonable by all group members.

If a group member fails to contribute sufficiently to the project during a week, then they should do an additional portion or task for the next week's project work. The distribution of the portions between group members should be agreed upon by the group members such that they both deem it fair. In the event where a group member has continuously failed to contribute sufficiently to an assignment, then the group may discuss the issue collectively and agree on an action plan to address the lack of contribution and the method for the non-contributing group member to make up the work that they've failed to complete.

2. Product description

- Start with a short and catchy project title (*not* "CS 362 project pitch") and your full names.

  Meal Match

  Ayush Baruah, Duncan Everson, Benjamin Kono, Aiden McCoy, Daniel Molina, Nicholas Shininger

- **Abstract:** The first paragraph of your document must be an abstract (or executive summary, or TL;DR) that explains your project at a high level. If someone read nothing but that one paragraph, what do you want them to know?

  The service Meal Match will allow users to be able to get meal information and assemble meal plans. Users will also be able to learn about the necessary ingredients to match those plans. Along with these meal plans, they will be able to see nearby stores which can sell them the required ingredients, be taught how to make those foods with provided recipes, and get nutrient information as to why those plans are best for them. The service will track their nutrient and meal consumption into a long term tracker to show their progress.

- **Goal:** What are you trying to do? For example, what task or problem will your system help users with?

  Make meal planning, prep, and long-term tracking easier. We will provide to users a single location where they can assemble their meal plans and get a list of ingredient location, price, and nutrient information for those meal plans. The meal and nutrient information will then be stored and retrievable by the user across usage sessions.

- **Current practice:** How is it done today, and what are the limits of current practice?

  Meal planning can be done through a service called Mealime, which offers custom meal planning, the ability to write a grocery list, and meal options based on dietary preferences. Mealime doesn't integrate with local stores though. Yummly provides many of the same features as Mealime, including meal recommendations and nutrient information. It does not have long term tracking or local ingredient information though. Nutrient and calorie tracking can be done through a service called MyFitnessPal, where the nutrient information that is shown is only for pre-entered meals. It does not have an ingredient list that it works off of, only previously entered nutrient information provided by other people.

- **Novelty:** What is new in your approach and why do you think it will be more successful than other approaches? Do not reinvent the wheel or reimplement something that already exists, unless your approach is different.

  The novelty of this project would be the combination of several services, rolled into one. While some of these approaches have been done before in separate applications/ services, none have been rolling them into one use for the consumer. The novelty is convenience, and an easy logical solution. Additionally, we would provide long-term meal information tracking, which Mealime, Yummly, and MyFitnessPal don't provide. We would also provide the local store interface to give users an actionable plan for gathering these ingredients, which Mealime and Yummly don't provide.

- **Effects:** Who cares? If you are successful, what difference will it make?

  People that utilize fitness tracking apps that are annoyed by the burden of using so many different apps will care about this implementation. Also, the people that want an actionable plan for gathering ingredients, and that want longer term nutrient and meal plan tracking will also care. Creating a localization of many different popular current practices will provide a centralized location for nutritional tracking and for meal prep, giving more people access to services that they may not be aware of in their pursuit of a single service. Additionally, adding the local store ingredient integration will allow people to get actionable plans for how to go about creating their meals. Also, adding long-term meal and nutrient tracking will help people to better understand their habits and better chart their progress toward any goals they may have.

- **Technical approach:** Briefly describe your proposed technical approach. This may include what system architecture, technologies, and tools you may use.

  We will use Node.JS's Express routing to serve React pages to users. We will use API calls to retrieve storefront information about available products from Kroger, Walmart, and Instacart. We will use Edamam to get nutrient information and meal recipes. We will use server-based infrastructure on local files to support the website's activity.

- **Risks:** What is the most serious challenge or risk you foresee when developing your project on time? How will you minimize or mitigate the risk? Don't state generic risks equally applicable to any project, like "we might run out of time".

  Our biggest risk is having limited access to the necessary APIs. This could include needing paid APIs, or being unable to find an API that gives the information we need correctly. A way of dealing with the lack of access to certain APIs and crucial information is to create systems where users can upload their own data into a collective database that we could then use as a new source of data for customers. Another risk is that the API will provide incorrect information to the user. A way of mitigating incorrect information would be to have a report system for the user to prompt the development team with inaccurate information given from an API.

- **Timeline:** Tentative high-level weekly goals for your project for the coming 9 weeks.

  Week 2: Elicit functional, non-functional, and external requirements for Meal Match.

  Week 3: Develop an understanding of the software architecture that Meal Match will entail

Week 4: Develop an understanding of the software design and coding guidelines that Meal Match will follow.

Week 5: Project architecture should be fully understood for Meal Match. Implementation for Meal Match should begin, following the software design and coding guidelines.

Week 6: Continue working on implementing Meal Match by working on its various features and trying to validate its functional requirements. Testing suites should begin being planned out for the early stages of implementation.

Week 7:(Project Implementation and Testing) Certain portions of Meal Match should be implemented, enough so testing can occur. Testing suites should be usable and verify real functional requirements.

Week 8: Polishing the code and combining various components of Meal Match should occur this week. More testing should be done in order to confirm a Minimum Viable Product and to prepare it for Beta Release.

Week 9: First prototype of the implementation should be completed, such that all requirements are satisfied.

Week 10: Incorporate feedback made regarding the first prototype. Finalize any changes made after the first prototype release, ensuring adequate modification to the documentation for those changes.

- 4+ major features you will implement.
    - Grocery Price Check for nearby stores
    - Search for grocery
    - Meal plan creator
    - Nutrient tracking system
        - Long term tracking
    - Recipe finder
- 2+ stretch goals you hope to implement.
    - Make the price check system a framework, not solely for grocery, but also for general shopping
    - Create a workout creation system to piece together workouts given the user has input what type of workout they want to do (ex. HIIT, Calisthenic, Weight Lifting, etc.).
        - Workout tracking

3. Use Cases (Functional Requirements)

1. **Find a Meal**

Actors: Meal creators (users)

Triggers: A meal creator wants to broaden their meal planning or making capabilities with a new meal's information.

Preconditions: A meal creator wants to incorporate a meal into their meal plan.

Postconditions: A meal creator has found and saved a meal that they are interested in into their list of saved meals.

List of Steps:

1. The meal creator browses a catalog of images and titles for meals they might try, and finds a meal that they are interested in.
2. The meal creator views the image, title, ingredients, and preparation instructions.
3. The meal creator adds the meal to their list of saved meals.

Extensions/Variations of Success Scenario:

Variation 1

1. The meal creator browses a catalog of images and titles for meals they might try, and finds a meal that they are interested in.
2. The meal creator views the image, title, ingredients, and preparation instructions.
3. The meal creator decides that they do not like that meal, and backs out to the meal catalog.
4. The meal creator selects a different meal from the catalog
5. The meal creator views the information about that meal.
6. The meal creator adds that meal to their list of saved meals.

Exceptions - Failure Conditions and Scenarios:

1. The meal creator views a list of images and titles for all of the meals available to them.

   Failure Condition 1:

   1. The meal creator browses the meal catalog and doesn't find a meal that interests them.
   2. The meal creator stops viewing the meal catalog by leaving the website.

2. The meal creator views the image, title, ingredients, and preparation instructions.

Failure Condition 2:

1. The meal creator views the information about the meal that they are interested in, but the meal is not something that they want to save.
2. The meal creator stops viewing the specific meal and leaves the website.

3. The meal creator adds the meal to their list of saved meals.

2. Macro and micro nutrient tracking system

***Actors*:**

- Registered user who wants a nutrition tracking application

***Triggers*:**

- User has created an account and is logged into their account.
- User selects the current day and logs their food with the quantity.
- User selects a previous day to log foods and/or look at nutritional information of that day.

***Preconditions*:**

- The user must have an account and be logged in
- The user must want a tracking system that can provide macro/micro nutrient information about current days and previous days if they logged foods for any past days.

***Postconditions*:**

- When a food is logged through a registered user, the system stores the logged food and its associated nutrient breakdown in the user's profile.
- The user can access and modify any previous day regardless if they previously inputted foods or not.
- The user can track their macronutrients and micronutrients for the current day

***List of Steps*:**

1. The user creates an account with an email and password
2. The user logs into their account with an associated email and password
3. The registered user clicks on the “Nutrition Tracking” page located on the navigation bar.
4. The logged in user searches for a food they want to input with the quantity.
5. The logged in user clicks on the day they want to input their food log.

6. The logged in user clicks on a specific time they want to log the food.
7. Once they have a food with a quantity and the specific time consumed, the user clicks "submit" or hits "enter" on their keyboard and the macro/micro nutrient information is added into that time slot.
8. Once they have imputed one or many foods, the logged in user clicks "submit" which will save that specific day and show the accumulated macro/micro nutrient information in a table format.

***Extensions/Variations of Success Scenario*:**

Variation 1: User Edits Previously Logged Data

- The user previously logged a food with the quantity and time consumed.
- The user wants to change previous logged data and can update the food, quantity, and/or time consumed.

Variation 2: User Adds a Food Not In Database

- The user searches for a food with the quantity
- If a specific food isn't provided by the database, the user has the option to add a name, quantity, and the macro/micro nutrients of a specific food.

***Exceptions - Failure Conditions and Scenarios*:**

Failure Condition 1: User Searches/Logs Food Without Quantity

- When a user attempts to log a food without quantity, the system will provide an error message: "Please provide all required information.", with the required input information in red.

Failure Condition 2: User Isn't Logged Into An Account

- If the user is trying to log information into the Nutrition Tracking System, the system will notify them "You must be logged in to access these features."

3. User wants to create a meal plan

Actors:

- User who has only recently started using Meal Match and wants to create their first meal plan

Triggers:

- User clicks on the "Meal Plan" section of Meal Match
- Meal Match sends a notification to the User to extend their current meal plan.

Preconditions:

- User must have an account
- User must be logged in

Postconditions:

- User has a working meal plan with meals scheduled in advance for them to take
- Meals have been portioned to include the correct nutrients and portions for the user

List of Steps:

1. Navigate to the Meal Plan section of Meal Match
2. User taps to create a new Meal Plan
3. User inputs various information about the Meal Plan
    - Purpose of Meal Plan (Required)
        - Lose Weight
            - Target Weight loss goal?
        - Build Muscle
        - Monitor Food Intake
    - Length of Meal Plan (Required)
        - Continuous?
        - Months?
        - Weeks?
    - Dietary Restrictions (Optional)
    - Cost Restrictions (Optional)
4. User hits submit
5. Meal Plan is created and User can access it in the Meal Plan section of Meal Match

***Extensions/Variations of Success Scenario*:**

- User includes the purpose and length of the meal plan but not dietary restrictions and cost restrictions. User is happy with the end Meal Plan
- User includes the purpose of the meal plan, length of the meal plan, and dietary restrictions, but not cost restrictions. User is happy with the end Meal Plan
- User includes the purpose of the meal plan, length of the meal plan, and cost restrictions but not dietary restrictions. User is happy with the end Meal Plan.

***Exceptions - Failure Conditions and Scenarios*:**

- User includes only the purpose of the meal plan but not the length, dietary restrictions, or cost restrictions. The system cannot generate a Meal Plan, leaving the user frustrated.
- User includes the purpose and length of the meal plan but does not include either dietary restrictions or cost restrictions. The Meal Plan generated includes meals that the user cannot eat, and they are dissatisfied with the outcome.
- User includes the purpose and cost restrictions but omits the length of the meal plan and dietary restrictions. The system defaults to a continuous Meal Plan, leaving the user frustrated.
- User includes the purpose, length, and dietary restrictions but omits cost restrictions. The Meal Plan exceeds their budget, causing them to abandon the plan.
- User wants to create a meal plan, but decides against it and backs out

4. **Creating an account**

Actors: New Users

Triggers: User, when faced with login screen, presses "Create an account"

Preconditions: The user does not have an account, or wants to create a new account

Postconditions: The user creates a new account after giving an email address and password.

List of Steps:

- After pressing the create account button, the user will be taken to a page to register.
- In this page, the user will give an email address and a password. The password will need to be typed twice identically to each other to confirm.
- Once the passwords are identical and a valid email address is given, the user will press "create account" and be brought to the home page.
- The passwords will need to satisfy a select quantity of conditions, including being larger than 8 characters long, less than 128 characters long, containing a number, and any combination containing only latin characters, numbers, special characters (@,#,$,%,etc.), dashes, and underscores.
- Account is placed into the listed accounts held as cookies on the website, used for the step below.
- When the user selects their account settings, if they go to the log out option, there will be an option to "select another account"- within this page this new account will be listed, along with another account they may have been logged into previously.

Extensions/Variations of Success Scenario:

- User gives valid email, ie. a string value that contains a string of characters, @, and a domain (.com, .net, etc)
- User gives valid password satisfying conditions listed in the steps category, and both password fields match identically.
- User presses “Create Account”
- The following conditions above being satisfied is considered a success scenario.

Exceptions - Failure Conditions and Scenarios:

- Should the email be seen as invalid/ blank, the website will display warning text of this.
- Should the password be invalid/ blank, the website will list the requirements of a password.
- Should the passwords not match, the website will say so.
- Any warning text will be removed from view once the condition that caused its appearances is removed

5. **User Stats/ tracking**

Actors: Registered users

Triggers:

- User, when logged into their account, navigates to their profile and selects stats.
- A user adds a new entry (a meal, workout, or any other relevant data point) that adds to their tracking metrics

Preconditions:

- User must have an existing account
- User must be logged in
- User must have existing data to view in stats. (For example, previous recipes/meals, previously logged activity.)

Postconditions:

- The user is able to view statistics and historical data about their interactions with Meal Match

List of Steps:

- User logs in to their existing account.
- User navigates to their profile
- From the profile user selects the stats tab
- System retrieves the users historical data (meal logs, nutrient information, workout details) and displays it to the user

- Graphs of relevant information (calorie consumed, weight lost, etc.) are also displayed

Extensions/Variations of Success Scenario:

- User successfully view their account statistics.

Exceptions - Failure Conditions and Scenarios:

- User is not logged in and so cannot navigate to the stats page.
- User does not have any existing data or history to view. (message is displayed to user)

6. **Check grocery prices at nearby stores**

***Actors*:** User

***Triggers*:** User wants to find the best grocery prices at nearby stores.

***Preconditions*:**

User has an account and is logged in

Users have a grocery list, know what items they need to buy, or find a meal on the app.

***Postconditions*:**

User receives a list of nearby stores with prices for the items.

User can compare prices and choose the best option.

***List of Steps*:**

1. User accesses the grocery price check feature in the app.
2. System asks the user to input their grocery list or select items from a meal plan they selected.
3. User inputs list or selects items.
4. System searches nearby stores that carry the items and retrieves price information.
5. System displays a list of the nearby stores with prices for each item.
6. User selects a store based on the price and location.
7. System provides directions to the selected store.

***Extensions/Variations of Success Scenario*:** User wants to modify the grocery list

3a1. User edits the grocery list.

3a2. System updates the search results with the revised list.

***Exceptions - Failure Conditions and Scenarios*:**

User does not have an account or is not logged in:
System prompts the user to log in or create an account.

System encounters an error retrieving price information:
System notifies the user of the error and suggests trying again or checking for updates.

No nearby stores carry the items on the grocery list:
System notifies the user that no nearby stores carry the items on their list, suggests alternative items or stores further away.

4. Non-functional Requirements

1. Accuracy - Accuracy is critical when calculating nutritional information and helping users make informed decisions based on the micro/macro nutrient information. Additionally, for the grocery price check feature, accuracy ensures that users receive reliable and up-to-date pricing information for nearby or online grocery stores.
2. Reliability - Reliability will make sure that the website is at least up 99.9% during the time it is operational. The system should be able to handle errors and include mechanisms to restore functionality within 1 hour in case of failures. Reliability will ensure that the website is available whenever the user needs it.
3. Usability - Usability will make sure Meal Match can provide a user-friendly UI. This will allow users to use the website and perform tasks without extensive tutorials or support. The main features should be accessible within three clicks from the homepage.

5. External Requirements

- Any non-”200” HTTP status code from an API call must be caught and handled by the server, such that the user is then presented a message and continued website functionality.
- The user should never encounter an error that takes away navigation or usage control of the website’s features.
- Features not related to storing information must be usable without an account.
- The service must be publicly available through a URL
- All code must be available to the public from GitHub, such that they can build out the project themselves and modify any aspect of the project.
- A README.md file must be present in the root directory, and include at least a brief description of the application, an installation guide, a usage guide, and information about what file to examine to find the documentation for each file’s code.
- All API interactions must have documentation regarding how to interact with them, and that documentation must be findable from the README.md, either directly, or through a path from the README.md to another documentation file that describes the API call process.

6. Team Process Description

Describe your quarter-long development process.

- Specify and justify the software toolset you will use.
    - Account Creation/Management

        - We will use MongoDB to store the users account information, as well as to store the nutrition and meal information inputted so that it stores information from previous days. We will use MongoDB because we have familiarity with it and it allows us to support a large amount of data.
    - Nutrition API
        - We will use Edamam API to support the macro/micro nutrient calculations and we will use it to search for recipes based on a search query. We will use Edamam API because it is fairly simple to implement and it is third-party tested which testifies to its accuracy.
    - Local Store APIs
        - We will use the Kroger, Walmart, and Instacart APIs to get information about the offerings of local stores in select regions. We will use a combination to reduce risk, so that if one doesn't work correctly, there will be others as well that we can grab from, and if one turns out to not fit well into our project, we can just drop it and focus on the others.
    - Frontend Implementation
        - We will use React for our frontend library. We will use react because we have previous experience with implementing it and it will make it easier to build components like charts or graphs for visualizing nutrition data and/or daily logs.

- Define and justify each team member's role: why does your team need this role filled, and why is a specific team member suited for this role?
    - Frontend Engineer/ UI design - Aiden McCoy and Benjamin Kono will be in charge of the frontend and the UI design. The frontend is an essential component to a complete web application because it is crucial that the user is able to easily navigate, use, and feel satisfied with the experience given by any web application. Aiden and Benjamin will be suited for this role because they both have experience in front-end development and love to make things visually engaging.
    - Backend Engineer
        - Ayush Baruah, Daniel Molina, and Nicholas Shininger will be in charge of the Backend Engineering. Backend Engineering is important to Meal Match's development because it holds all the internal structure of the system and holds the data as well. Ayush Baruah, Daniel Molina, and Nicholas Shininger will be responsible for this portion of the project due to our experience with backend development in other projects.
    - Tester
        - Nicholas Shininger and Ayush Baruah will be in charge of the Testing portions of the development of Meal Match. This is an essential part of its development since it is the portion where we validate and verify the actual

        - components and architecture of Meal Match. Both Nicholas and Ayush have volunteered to be the Testers since they both have various levels of experience working on testing scripts and suites for components.
    - Full-stack Engineer
        - Duncan Everson will be in charge of taking the role of Full Stack Engineer. A Full Stack Engineer is vital for development because they understand how to do all aspects of the project and where difficulties and opportunities may arise during the development of Meal Match. This can help with decision-making during the SDLC in order to increase the feasibility and drop ideas that would distract us from our goal or increase complexity unnecessarily. Duncan is a good person to take the role of Full Stack Engineer because he has some experience with full stack development of his own during previous work.
    - Backend UI Engineer
        - Nicholas Shininger will aid the frontend team with implementing the JavaScript functions inside of the React pages to aid in data transfer between the frontend and backend. This is a vital position as it is important to have someone that is familiar with both the sending and retrieving functionality of the data transfers to be able to ensure the correct data is both send and received with the React pages and the server. Nicholas is a good choice for taking on this role as he feels confident in being able to bridge the gap between frontend and backend.
- Provide a schedule for each member (or sub-group) with a *measurable*, concrete milestone for each week. "Feature 90% done" is not measurable, but "use case 1 works, without any error handling" is.

Sub-Group Timeline

Week 4 - Start Architecture / Design Specs, Finish With Implementation / Documentation

- Frontend/UI design
    - Complete figma wireframe for the website
    - Complete figma prototypes for the pages
- Backend
    - Create a basic javascript server application, and stub out express routes.
- Testing
    - N/A
- Backend UI Engineer
    - N/A

Week 5 - Implementation / Documentation

- Frontend/UI design

        - Begin work on the React pages. Each use case should work correctly without any error handling.
    - Backend
        - Set up database to handle storing and retrieving user data
        - Integrate user data schema with database storage and retrieval
        - Set up all API interactions for data retrieval
    - Testing
        - Evaluate and compare the requirements to their functionality in the page prototypes
            - Evaluate what necessary error handling will need to exist for both frontend and backend.
            - Begin writing tests
    - Backend UI Engineer
        - Document what information needs to be sent to and received from the React pages.

Week 6 - Continue Implementation / Documentation, Finish With Large-Scale Testing

- Frontend/UI design
    - Continue work on React pages. Each use case should fully work with adequate error handling and input sanitization on the front end
- Backend
    - Fill out route logic for handling the sent data to and from the React pages. Usage cases should work with error handling on the back end.
- Testing
    - N/A
- Backend UI Engineer
    - N/A

Week 7 - Start Large-Scale Testing, Finish With Beta Release

- Frontend/UI design
    - N/A
- Backend
    - Finish Documentation for all code
- Testing
    - Test application's implementation of use cases against the specified requirements for those cases
- Backend UI Engineer
    - N/A

Week 8 - Start With Beta Release, Finish With Final Release

- Frontend/UI design

        - Adjust design based on user feedback as possible.
    - Backend
        - Receive any bug reports from testing team and correct them
    - Testing
        - Receive any bug feedback from users and test those bugs
            - Send details about any complex bugs to Backend team
    - Backend UI Engineer
        - N/A

- Specify and explain at least three major risks that could prevent you from completing your project.
    - The API we are using to find prices for nearby stores and/or online stores doesn't have the price of a certain item.
    - Using various APIs can increase the complexity of our project and certain APIs won't be able to easily communicate or work with each other without serious help needed from the developers to make them work.
    - Having multiple API calls can slow down the App in its totality, making running our app slow.
- Describe at what point in your process external feedback (i.e., feedback from outside your project team, including the project manager) will be most useful and how you will get that feedback.
    - We will get feedback by at least one other group in our CS362 course by both the presentation feedback and from a more detailed conversation communicating what they liked and disliked about the project. We will use this feedback during the third, fifth, and eighth weeks of the term. In addition to feedback from one other group, we will use the feedback given to us by our project manager each week we meet with them. We will use the feedback to make changes to our project and ensure we are meeting our user expectations. The most useful feedback given to us will be feedback that focuses on how to improve the usability, functionality, and overall user experience of our project.

i. Risk assessment

1. **Limited Access to Necessary APIs**
    - Likelihood: Medium
    - Impact: High
    - Evidence: Many grocery and nutrition APIs require paid subscriptions, rate limits, or strict terms of use.
    - Steps to Reduce Likelihood/Impact:
        - Find alternative free or affordable APIs.

        - Build a flexible system where users can manually input data when API data is unavailable.
    - Plan for Detecting the Problem:
        - Test API access early and check documentation for pricing, limitations, and accuracy.
        - Implement logging to track API failures or incorrect responses.
    - Mitigation Plan:
        - Allow user-generated data to fill gaps.
        - Switch to a different API if our current choices become unavailable or too expensive.
        - Provide users with disclaimers about data reliability.
    - Changes from Requirements Document:
        - More specific mitigation steps, such as adding user-generated data and logging API failures.

2. **Dependency on Third-Party APIs and Services**
    - Likelihood: Low
    - Impact: High
    - Evidence: If a key third-party API becomes unavailable or changes pricing, the app's functionality may be affected.
    - Steps to Reduce Likelihood/Impact:
        - Identify multiple API options for each type of data.
    - Plan for Detecting the Problem:
        - Monitor API response times and failure rates.
        - Set up alerts for API outages or changes in pricing models.
    - Mitigation Plan:
        - Design APIs in a modular way so replacements can be swapped.
    - Changes from Requirements Document:
        - Originally focused on API selection, now also planning for potential API failures.

3. **API Returns Incorrect or Outdated Data**
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: Some grocery APIs may not update frequently, leading to incorrect pricing or availability. Nutrition APIs may not have consistent data across sources.
    - Steps to Reduce Likelihood/Impact:
        - Check data from multiple sources when possible.
        - Allow users to report incorrect data.

- Plan for Detecting the Problem:
    - Implement logging for unusual data changes.
- Mitigation Plan:
    - If data is unreliable, display the estimated ranges of exact values.
    - Offer alternative data sources (e.g., user-reported prices or manual entries).
- Changes from Requirements Document:
    - Added more mitigation steps and implemented logging for unusual data changes.

4. **Performance and Scalability Issues**
    - Likelihood: Medium
    - Impact: Medium to High
    - Evidence: If the app has many users, frequent API calls and database queries could slow performance. Poorly optimized queries could lead to long loading times.
    - Steps to Reduce Likelihood/Impact:
        - Implement API rate-limiting and request batching.
        - Optimize database queries with indexing and caching.
    - Plan for Detecting the Problem:
        - Monitor system performance using logging and profiling tools.
        - Conduct load testing to identify bottlenecks.
    - Mitigation Plan:
        - Scale the backend with cloud services or database replication if needed.
        - Reduce reliance on real-time API calls by caching frequently accessed data.
    - Changes from Requirements Document:
        - Was not originally addressed, now included due to high potential data loads.

5. **Difficulties in Integrating Multiple Data Sources**
    - Likelihood: Medium
    - Impact: Medium
    - Evidence: Different APIs may return data in inconsistent formats, requiring extra processing. Some APIs could lack documentation, making integration difficult.
    - Steps to Reduce Likelihood/Impact:
        - Standardize data handling with common response structures.
        - Test API integration early in development.
    - Plan for Detecting the Problem:

  - Run automated tests to verify that API data maps correctly to our database schema.
  - Implement logging to track failed API responses.
- Mitigation Plan:
  - Write data transformation scripts to handle inputs.
- Changes from Requirements Document:
  - Originally focused on API access, now addressing data inconsistencies across multiple APIs.

ii. Project schedule

Project Roadmap Week 5 (2/3/25-2/9/25)

Task	Team	Status	Reliant Task	Date	Notes
1 - Project Architecture / Design	All	Ind…		2/4/25	Fill out all bullet points related to the Project Architecture and Design Canvas page
2 - Prototypes	Frontend	Ind…		2/4/25	Complete development of the page prototypes
3 - 'Homepage' HTML	Frontend	Rel…	W5-2	2/8/25	Develop the basic HTML structure for the 'Homepage' React page design from Figma
4 - 'Help Page' HTML	Frontend	Rel…	W5-2	2/8/25	Develop the basic HTML structure for the 'Help Page' React page design from Figma
5 - 'Food/Food Diary' HTML	Frontend	Rel…	W5-2	2/8/25	Develop the basic HTML structure for the 'food/food diary' React page design from Figma
6 - 'Grocery/Grocery List' HTML	Frontend	Rel…	W5-2	2/8/25	Develop the basic HTML structure for the 'Grocery/Grocery List' React page design from Figma
7 - Create CSS styling rule set for pages	Frontend	Rel…	W5-2	2/8/25	Create / assemble CSS styles for the documents

Project Roadmap Week 5 (2/3/25-2/9/25)

Task	Team	Status	Reliant Task	Date	Notes
8 - Database Framework with MongoDB	Backend	Ind…		2/8/25	Create the database interaction account and grab an access key to interact with the DB
9 - MongoDB Schema Model Files	Backend	Ind…		2/8/25	Create schemas for Meals, Ingredient Providers, and Users
10 - API Interactions	Backend	Ind…		2/8/25	Set up API interactions for sending HTTP requests to Edamam, Kroger API, Walmart API, and InstaCart API.
11 - Mid-Term Project Presentation and Report	All	Rel…	W5-1 W5-2	2/9/25	Complete the Mid-Term Project Presentation and Report assignment such that the deliverable satisfies the conditions of the given rubric.

Project Roadmap Week 6 (2/10/25-2/16/25)

Task	Team	Status	Reliant Task	Date	Notes
1 - ‘Account Creation’ HTML	Frontend	Rel…	W5-2	2/14/25	Develop the basic HTML structure for the ‘Account Creation’ React page design from Figma
2 - ‘Login’ HTML	Frontend	Rel…	W5-2	2/14/25	Develop the basic HTML structure for the ‘Login’ React page design from Figma
3 - ‘Food/Recipe Search’ HTML	Frontend	Rel…	W5-2	2/14/25	Develop the basic HTML structure for the ‘Food/Recipe Search’ React page design from Figma
4 -	Frontend	Rel…	W5-2	2/14/25	Develop the basic HTML structure

Project Roadmap Week 6 (2/10/25-2/16/25)

Task	Team	Status	Reliant Task	Date	Notes
'Food/Meal Plan Creation' HTML					for the 'Food/Meal Plan Creation' React page design from Figma
5- 'Grocery/Search Groceries' HTML	Frontend	Rel...	W5-2	2/14/25	Develop the basic HTML structure for the 'Grocery/Search Groceries' React page design from Figma
6 - 'Homepage' Frontend JS	Frontend/ Backend UI	Rel...	W5-3	2/16/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Homepage.
7 - 'Help Page' Frontend JS	Frontend/ Backend UI	Rel...	W5-4	2/16/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Help page.
8 - 'Food/Food Diary' Frontend JS	Frontend/ Backend UI	Rel...	W5-5	2/16/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Food/Food Diary page
9 - 'Grocery/ Grocery List' Frontend JS	Frontend/ Backend UI	Rel...	W5-6	2/16/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Grocery/Grocery List page
10 - 'Homepage' Routes	Backend	Rel...	W5-3	2/16/25	Handle information reception from the frontend homepage. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
11 - 'Help Page' Routes	Backend	Rel...	W5-4	2/16/25	Handle information reception from the frontend help page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.

Project Roadmap Week 6 (2/10/25-2/16/25)

Task	Team	Status	Reliant Task	Date	Notes
12 - ‘Food/Food Diary’ Routes	Backend	Rel…	W5-5	2/16/25	Handle information reception from the frontend Food/Food Diary page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
13 - ‘Grocery/ Grocery List’ Routes	Backend	Rel…	W5-6	2/16/25	Handle information reception from the frontend Grocery/Grocery List page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.

Project Roadmap Week 7 (2/17/25-2/23/25)

Task	Team	Status	Reliant Task	Date	Notes
1 - Project Implementation and Documentation	All	Ind…		2/18/25	Fill out all bullet points related to the Project Implementation and Documentation Canvas page
2 - Testing and CI	All	Ind…		2/21/25	Address all bullet points and requirements in the Testing and Continuous Integration Canvas page within the living document
3 - ‘Nutrition/Day’ HTML	Frontend	Rel…	W5-2	2/22/25	Develop the basic HTML structure for the ‘Nutrition/Day’ React page design from Figma
4 - ‘Nutrition/ History’ HTML	Frontend	Rel…	W5-2	2/22/25	Develop the basic HTML structure for the ‘Nutrition/History React page design from Figma
5 - ‘Account	Frontend/	Rel…	W6-1	2/23/25	Make the JavaScript functions in

Project Roadmap Week 7 (2/17/25-2/23/25)					
Task	Team	Status	Reliant Task	Date	Notes
Creation' Frontend JS	Backend UI				the React file for handling data transfers and user requests from the Account Creation page
6 - 'Login' Frontend JS	Frontend/ Backend UI	Rel…	W6-2		Make the JavaScript functions in the React file for handling data transfers and user requests from the Login page
7 - 'Food/Recipe Search' Frontend JS	Frontend/ Backend UI	Rel…	W6-3	2/8/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Food/Recipe Search page
8 - 'Food/Meal Plan Creation' Frontend JS	Frontend/ Backend UI	Rel…	W6-4	2/8/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Food/Meal Plan Creation page
9 - 'Grocery/ Search Groceries' Frontend JS	Frontend/ Backend UI	Rel…	W6-5		Make the JavaScript functions in the React file for handling data transfers and user requests from the Grocery/Search Groceries page
10 - 'Account Creation' Routes	Backend	Rel…	W6-1	2/8/25	Handle information reception from the frontend Account Creation page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
11 - 'Login' Routes	Backend	Rel…	W6-2		Handle information reception from the frontend Login page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
12 - 'Food/Recipe	Backend	Rel…	W6-3	2/8/25	Handle information reception from the frontend Food/Recipe

Project Roadmap Week 7 (2/17/25-2/23/25)

Task	Team	Status	Reliant Task	Date	Notes
Search' Routes					Search page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
11 - 'Food/Meal Plan Creation' Routes	Backend	Rel…	W6-4	2/8/25	Handle information reception from the frontend Food/Meal Plan Creation page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.

Project Roadmap Week 8 (2/24/25-3/2/25)

Task	Team	Status	Reliant Task	Date	Notes
1 - 'Nutrition/Day' Frontend JS	Frontend/ Backend UI	Rel…	W7-3	2/28/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Nutrition/Day page
2 - 'Nutrition/History' Frontend JS	Frontend/ Backend UI	Rel…	W7-4	2/28/25	Make the JavaScript functions in the React file for handling data transfers and user requests from the Nutrition/History page
3 - 'Grocery/Search Groceries' Routes	Backend	Rel…	W6-5	3/2/25	Handle information reception from the frontend Grocery/Search Groceries page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.

Project Roadmap Week 8 (2/24/25-3/2/25)

Task	Team	Status	Reliant Task	Date	Notes
4 - 'Nutrition/Day' Routes	Backend	Rel…	W7-3	3/2/25	Handle information reception from the frontend Nutrition/Day page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.
5 - 'Nutrition/History' Routes	Backend	Rel…	W7-4	3/2/25	Handle information reception from the frontend Nutrition/History page. Additionally, serve correct frontend React pages with correct information to the user, based on received requests.

Project Roadmap Week 9 (3/3/25-3/9/25)

Task	Team	Status	Reliant Task	Date	Notes
1 - Beta Release	All	Ind…		3/4/25	All major components work, Project is ready for feedback
2 - Beta Demo Presentation		Rel…	W9-1	3/5/25	Create a presentation that addresses all information from the Demo Presentation Instructions Canvas page
3 - Address Issues/ Suggestions	All	Rel…	W9-1	3/9/25	Take feedback and either make adjustments based on it, or leave it be.
4 - Final Release	All	Ind…		3/9/25	Finish all project work

Project Roadmap Week 10 (3/10/25-3/16/25)					
Task	Team	Status	Reliant Task	Date	Notes
1 - Final Demo Presentation	All	Rel…	W9-4	3/10/25 3/12/25	Create a presentation that addresses all information from the Demo Presentation Instructions Canvas page

iii. Team structure

Ayush Baruah, Backend Engineer / Tester

Duncan Everson, Full-stack Engineer / Documentation Verification

Benjamin Kono, Engineer in charge of UI design / Frontend Engineer

Aiden McCoy, Engineer in charge of UI design / Frontend Engineer

Daniel Molina, Backend Engineer

Nicholas Shininger, Tester / Backend Engineer / Backend UI Engineer

Frontend Engineer:

Deal with React pages, filling out both the HTML portions and the JavaScript that is inside of the React documents. Also responsible for developing the CSS rules.

Backend Engineer:

Deal with the logic of the server catching of requests and the sending of the appropriate pages and data through express routes. Also deals with server database interactions.

Tester Engineer:

Execute test cases intended to break the application from a user's perspective.

Assemble and run a testing suite to test JavaScript backend functionality.

Backend UI Engineer:

Ensures JavaScript functions in both React pages and Server interact correctly in their requests and information transfers.

iv. Test plan & bugs

1. Testing plans:
    a. Unit Testing - Jest, JUnit, Supertest - React Testing Library

        i. Unit Testing is going to be done first to isolate small pieces of code and make sure that certain components of the application are working and are done properly by themselves
            1. Backend
                a. Jest
                b. Supertest
            2. Frontend
                a. React Testing Library
                b. Jest
        ii. While not holding its category, we are also going to be doing API testing within this portion. We are going to test the API logic and make sure that it works.
            1. API Testing
                a. Supertest
    b. Integration Testing - Jest, MongoDB Memory Server
        i. Integration testing is going to be the next phase of testing. We are going to integrate all the portions that were within our unit testing suites and combine them to work with each other as a cohesive single-unit
            1. Jest
            2. MongoDB Memory Server
    c. End-to-End Testing - Cypress
        i. E2E testing will be used to simulate proper user flow test full capabilities and go through proper test scenarios that were created for our user and integration tests.
            1. Cypress
    d. Performance Testing - K6, Lighthouse
        i. Performance testing will be done in order to ensure that the application can work properly under stress and works efficiently and quickly
            1. K6 (Load testing)
            2. Lighthouse (frontend performance)

All bugs will be reported and handled through Github issues and will be systematically debugged. This will ensure proper logging and tracking of bugs and create a system to track which bugs have been debugged, which are regressive, and which are yet to be fixed.

v. Documentation plan

Top-level documentation:

1. User Guide - Project Overview, Node Deployment Instructions, Bug Report Instructions, Known Bugs
2. Developer Guide - Architecture Overview, Components Overview, APIs used (also describes HTTP request structure for each), MongoDB Schema Documentation, File

Structure, and Links to Style Guides, How to obtain the source code, How to test and add new tests, and how to build a release of the software.

In-file structure documentation:

3. Testing Documentation - JSDoc Documentation for Testing Suite
4. Front End Code Documentation - React JSDoc documentation practice, CSS description comments and sectioning for groups of rules, HTML comment descriptions for sections/divs.
5. Back End Code Documentation - JSDoc documentation practice

7. Software Architecture

Components

1. Meal Management System
   Functionality: A system designed to help users create meal plans, generate grocery lists, and track nutritional intake.
   Interface: Users interact through an interface to input preferences, view plans, and track their meals and nutrition.
2. Meal Plan Generator
   Functionality: Generates personalized meal plans based on dietary preferences, goals, and restrictions.
   Interface: Users input dietary goals and restrictions, and the system displays meal plans with recipes and nutritional breakdowns.
3. Grocery List Generator
   Functionality: Converts the meal plan into a categorized shopping list of ingredients, helping users organize and track what to buy.
   Interface: Users view and interact with the generated list, marking off purchased items and adding custom items as needed.
4. Nutrient and Macro Tracking
   Functionality: Tracks users' intake of calories, proteins, fats, and carbohydrates, providing feedback on how their diet aligns with their goals.
   Interface: Users log their meals, and the system provides a breakdown of nutritional intake.
5. User Interface, Through React Webpages
   Functionality: The interface where users interact with all of the software components.
   Interface: Web Application, where users will view displayed react pages that have specific and select information related to the usage cases.
6. Database Interaction
   Functionality: Stores and organizes all important data related to the user data schema.

Interface: Utilization of MongoDB and the mongoose node.js dependency for backend access of stored information for displaying it to the react pages.

7. External API Interactions

   Functionality: Fetches external data from outside sources that is needed to satisfy the use cases.
   Interface: Utilization of Fetch API node.js dependency to send HTTP requests from the server to the Edamam, Kroger, Walmart, and InstaCart APIs.

- Describe in detail what **data** your system stores, and how. If it uses a database, give the high level database schema. If not, describe how you are storing the data and its organization.

We will be storing User Account, Meal, Ingredient Provider, and Nutrition Data. We will use MongoDB to store the data. This storage will entail the following schemas:

```
Meal Schema
{
        meal_id: Int,
        name: String,
        img: String,
        recipe: String,
        ingredients: String,
        nutrition: {
                        calories: Int,
                        carbs: Int,
                        protein: Int,
                        fat: Int,
                        dietary_restrictions: [String], // e.g., ["vegan", "gluten-free"]
                    },
}

Ingredient_Provider Schema
{
        ing_provider_id: Int,
        ingredient_name: String,
        provider_name: String,
        provider_location:
        price: Double,
        date_posted: Date
        date_updated: Date
}

User Schema
{
        user_id: Int,
        name: String,
        password: String,
        preferences: {
                        dietary_restrictions: [String], // e.g., ["vegan", "gluten-free"]
                        favorite_meals: [String], // e.g., ["Italian", "Mexican"]
                        caloric_goal: Number, // User's daily caloric target
                        macro_split: {
                                        protein: Int,
                                        carbs: Int,
                                        fats: Int
                                       }
                      },
        saved_meals: [Meals],
        meal_plans: [[Meals]],
        nutrition_log: [[Meals]],
        created_at: Date,
        updated_at: Date
}
```

- If there are particular **assumptions** underpinning your chosen architecture, identify and describe them.

We assumed that users will spend a significant amount of time in each usage session. This contributed to our decision to make this a web application, as we think that users that engage in prolonged work would prefer to engage the work through their computers rather than their phones.

We assumed that users will want a meal and nutritional history for themselves, and that they will want to view it repeatedly.

We assumed that users will want to adjust and modify their meal plans, and create multiple meal plans, all simultaneously saved to their account.

We assumed that users would want to access the data that they assemble between usage sessions, and that they would also want to access a pool of saved user entered data. This contributed to our decision to utilize MongoDB as it is a storage option that has a free tier that we assumed would be more than enough for a small live web application.

We also assumed that we did not need a traditionally relational database, as we don’t have normalized relationships between data, and also because many of our data values are nested. This nested structure makes a document based DB like mongoDB a better fit.

- **For each of two decisions pertaining to your software architecture**, identify and briefly describe an **alternative**. For each of the two alternatives, discuss its pros and cons compared to your choice.

One alternative for our software architecture’s Database Interaction component was to implement a relational database. We decided to implement a document based DB using the MongoDB service. One pro of a relational DB compared to a document based DB is that it would allow us to store our data in stricter entity tables with absolutely defined attributes and relationships compared to one another. This strict structure guarantees consistency between entries, which makes it easier to manage complex systems. A con of a relational DB compared to a document based DB is its lack of flexibility in handling complex, nested, and unstructured data. This means relational DBs can be challenging to work with when we have evolving data types.

An alternative for our software architecture’s User Interface being through a web application, was having it exist in a desktop application. One pro of desktop applications compared to web applications is that applications made for desktop downloads and that run locally on a machine can be developed to have vastly better performance than web applications. This is significant as we plan to be storing data related to the users that will be retrievable by the users. One con is that it can only be run locally on one machine. This limits the availability of the service when users are on the go, unless they have a laptop. Another con of desktop applications is that users need to do manual installation of the service, and manual updates. This makes pushing developments in the service to users require more coordination between users and the developers, as users must make the decision to improve their version of the application, as opposed to it automatically being improved in a web application.

8. Software Design

Meal Management System:
Packages/Classes:

- MealManager: Handles meal planning, grocery list generation, and nutritional tracking.
- UserPreference: Stores user-specific dietary restrictions and goals.
- MealPlan: Represents a structured meal plan.

Responsibilities:

- Coordinates interactions between meal planning, grocery lists, and nutrition tracking.

- Manges user preferences to ensure personalized meal recommendations.
- Integrates with the database and external APIs for fetching relevant meal and nutrition data.

Meal Plan Generator:
Packages/Classes:

- MealPlanGenerator: Creates meal plans based on user preferences and stored meal data.
- Meal: Stores meal details like name, ingredients, recipe, and nutrition facts.
- RecipeFetcher: Retrieves recipe information from the database or external sources.

Responsibilities:

- Takes user dietary preferences and goals to generate a meal plan.
- Pulls meal and ingredient data from the database.
- Suggests recipes based on user preferences and available ingredients.

Grocery List Generator:
Packages/Classes:

- GroceryListManager: Creates and manages grocery lists.
- Ingredient: Represents ingredients, including quantity and store availability.
- PriceFetcher: Queries prices from external grocery store APIs.

Responsibilities:

- Converts meal plans into structured grocery lists.
- Organizes items by category for easier shopping.
- Integrates with external APIs to fetch real-time grocery prices.

Nutrient and Macro Tracking:
Packages/Classes:

- NutrientTracker: Logs and analyzes user meal consumption.
- UserLog: Stores historical nutrition intake.
- ProgressVisualizer: Generates reports for long-term tracking.

Responsibilities:

- Records daily food intake and calculates macronutrient breakdown.

- Tracks progress toward user-defined dietary goals.
- Provides insights on long-term dietary trends.

User Interface (React Web App):
Packages/Classes:

- MealPlanPage: Displays generated meal plans.
- GroceryListPage: Allows users to interact with their shopping list.
- NutritionDashboard: Provides a visual representation of nutrient intake.

Responsibilities:

- Serves as the frontend for user interaction.
- Displays dynamically updated meal plans, grocery lists, and nutrition tracking.
- Sends and receives data from the backend through API calls.

Database Interaction:
Packages/Classes:

- UserModel: Defines user-related data and preferences.
- MealModel: Stores meal details, including ingredients and nutrition.
- IngredientProviderModel: Manages grocery item availability and pricing.

Responsibilities:

- Stores and retrieves structured data efficiently.
- Maintains user profiles, saved meals, and grocery provider information.
- Ensures data consistency and indexing for fast queries.

External API Interactions:
Packages/Classes:

- APIClient: Handles HTTP requests to third-party APIs.
- Edamam Service: Fetches nutritional data for ingredients and meals.
- GroceryStoreService: Interfaces with Kroger, Walmart, and Instacart for grocery prices.

Responsibilities:

- Fetches nutritional data for user meals.
- Retrieves grocery prices and availability in real-time.

- Ensures secure and efficient API communication.

9. Coding Guidelines

HTML / CSS: Either Google style guide: https://google.github.io/styleguide/htmlcssguide.html
JavaScript: AirBnB JavaScript Guideline - https://github.com/airbnb/javascript