Manger Design Document

A comprehensive solution to managing food

Luke Williams

Contents

Section 1: Objectives

Manger is a project that is a complete solution for food planning. It will contain the following main features:

- **A Food Inventory Manager** - to manage the supply of food and prevent it going out of date and having to be thrown away.
- **Meal Planner** - to plan what meals are to be eaten when, shown in a timetable.
- **Recipe Recommendation** - to suggest recipes to use based on the existing inventory of food, cost, simplicity, preferred diet, and preference based on previous recipes.
- **Recipe Guide** - A simple step-by-step instruction guide with interactive timers and multitasking to help guide the user through the cooking process.
- **Shopping List** - Given a meal plan or list of meals, the ability to instantly generate shopping lists to either take to the supermarket or transfer into an online delivery platform and add to the inventory as you tick the food off.

All 5 of these features generally have their own independent applications, but there are no solutions that take into account all 5 of these independent features.

This application may provide many different benefits:

1. **Food Waste** - as ingredients are forgotten about and left behind.
2. **Meal Repetition** - eating the same meal over and over again based out of comfort.
3. **Missed Meals** - suddenly realising that there aren’t the ingredients for the meals and not being able to cook them.
4. **Meal Stress** - not being sure what to do or getting confused when it comes to effectively balancing multiple actions at the same time.

Section 2: Requirements

1. **Food Inventory**
   a. Adding and removing food from the inventory
2. **Planning**
   a. Recommend meals based on existing inventory, cost, simplicity, and preference.
   b. Show the user what ingredients it would use that they already have
   c. Create meal plans that can be reused
   d. Create instant shopping lists based off the meal plans that can be added to the food inventory when purchased. e. Show a meal timetable for what meals can be had on what date (printable).
3. **Meal Preparation**
   a. Being able to add your own meals using “meal flow”, which is a representation of recipes that are interactive and help with multi-tasking
   b. Being able to use “meal flow” when cooking to help guide you throughout the cooking process.

Section 3: User Profiles

Chapter 3.1: The Family of 4

- Not much time on their hands.
- Having to manage a large amount of food.

How this technology helps them:

- They are able to manage the large amount of food and use meal plans that use food that may be soon out of date.
- They are able to generate shopping lists without having to manually write everything down.
- They can re-use meal plans so they don’t have to start from scratch each time.

Chapter 3.2: The Student

- Living on their own for the first time
- Not confident on how to cook.
- Isn’t as adept at coming up with meals based on existing ingredients.

How this technology helps them:

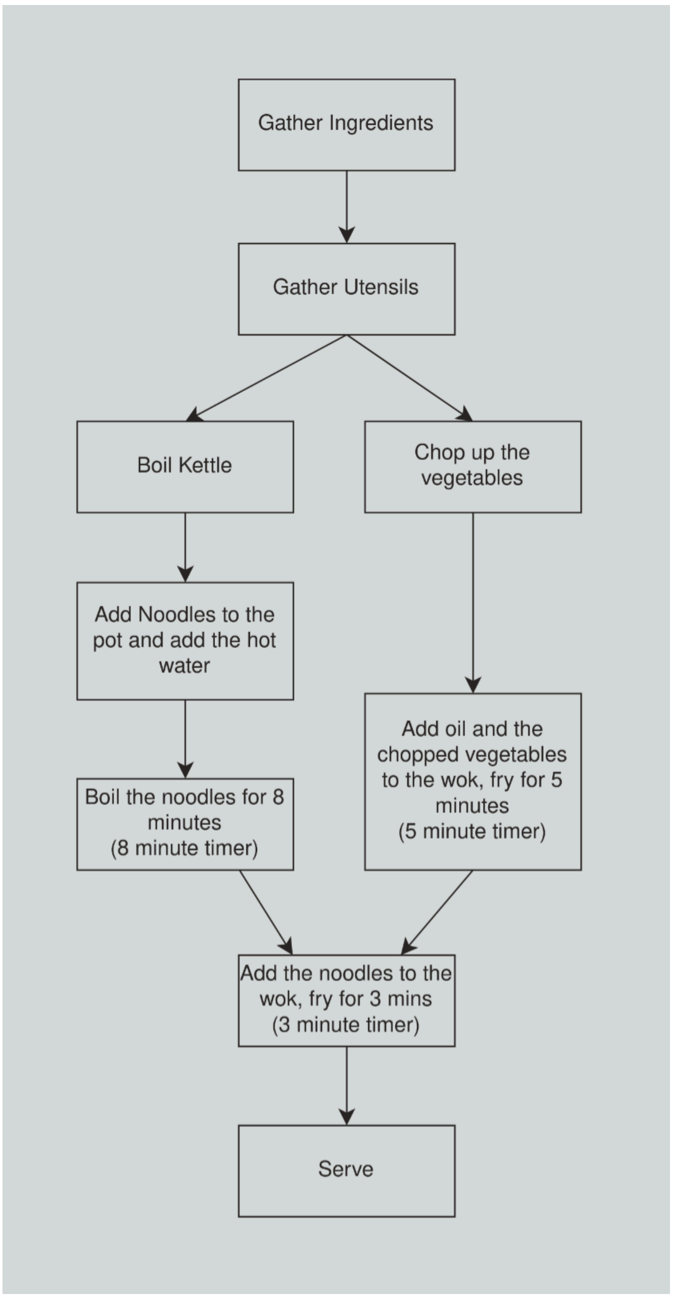
- They can access a wide range of recipes that they may like, and can give feedback to the system to help suggest meals to use.

Section 4: Design

Chapter 4.1: Meal Flow

In order to implement the requirement of having interactive meal plans, there needs to be some way of representing meals that allows for multitasking and timer functionality to be presented as and when the user needs.

4.1.1: Example


Gather Ingredients
Gather Utensils
Boil Kettle
Chop up the vegetables
Add Noodles to the pot and add the hot water
Add oil and the chopped vegetables to the wok, fry for 5 minutes (5 minute timer)
Boil the noodles for 8 minutes (8 minute timer)
Add the noodles to the wok, fry for 3 mins (3 minute timer)
Serve

To the right is an example of a recipe in meal-flow form. The arrows show the "flow" of the recipe. Tasks where the arrow originates from are dependencies of tasks where the arrow points to.

For example, you have to boil the kettle before adding the noodles and hot water to the pot.

Where two arrows branch off is where multitasking can be used.

For example, you can boil the kettle and noodles while chopping up the vegetables, which saves time overall.

Using this representation of a recipe, the user can navigate step by step through the instructions as they cook, and also keep track of everything that is happening through the use of handy timers.

A split-screen can be used to show the case where two actions are happening at the same time (like the kettle being boiled while the vegetables are chopped).

This representation can also easily be stored on a computer, due to it being an implementation of the graph data structure.

4.1.2: Storage in JSON

Representing these meal flows can be done in JSON.

See below an example of the above meal being represented in JSON

```
[
  {command: "Gather ingredients", timer: null, next: [1]}, // step 0
  {command: "Gather utensils", timer: null, next: [2, 5]}, // step 1
  {command: "Boil the kettle", timer: null, next: [3]}, // step 2
  {command: "Add noodles and water to the pot", timer: null, next: [4]}, // step 3
  {command: "Boil the noodles for 8 mins", timer: 8, next: [7]}, // step 4
  {command: "Chop up the vegetables", timer: null, next: [6]}, // step 5
  {command: "Fry the oil and veg for 5 minutes", timer: 5, next: [7]}, // step 6
  {command: "Add the noodles, fry for 3 minutes", timer: 3, next: [8]}, // step 7
  {command: "Serve", timer: null, next: []}, // step 8
]
```

This list of instructions contains the command, the timer to display (if applicable) and the next steps after completing that instruction.

This design allows easy jumping from instruction to instruction as the recipe progresses.

The limitation of this implementation is that when proceeding to the next instruction, it must be confirmed that all instructions that depend on that instruction are completed before progressing.

For example, perhaps the vegetables fry quicker than anticipated and ultimately the user reaches the stage where noodles and vegetables need to be combined, but they are still waiting on the noodles. At this point, when the user tries to progress after frying the vegetables, the split screen should move to a single page focussing on the noodles being waited for, before the user is shown the combination step.

This limitation can be circumvented through checking each time the user presses next as to whether the instruction we are progressing to has any other dependencies, and if those dependencies have been completed. Therefore we must store a list of instructions that have been completed.

Chapter 4.2: Databases

This will document how data in the application will be stored, which will be in a relational database.

4.2.1: Models

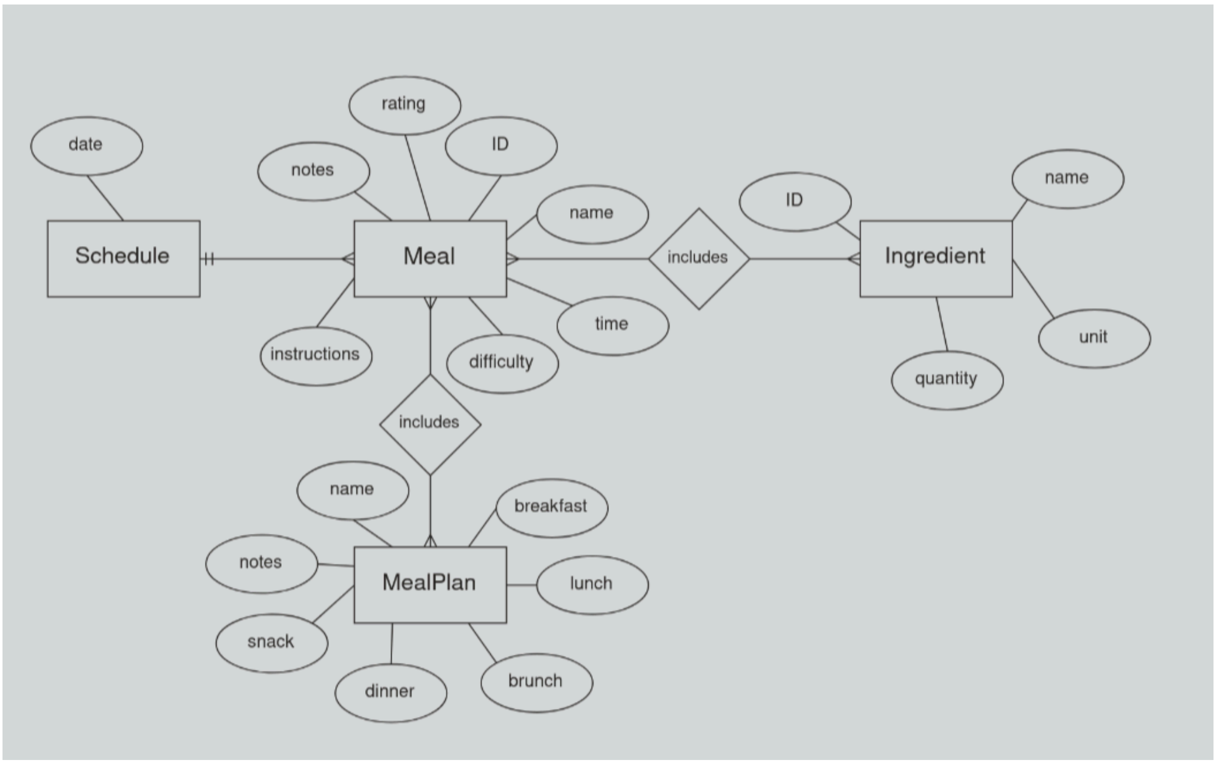
The models are the different objects that we want to store in the database. They will have attributes, which are data about the model that we want to store.

I have created a list of models and underneath each model, the attributes that the model will store:

- **Meal**
  - ID
  - name
  - estimated time
  - difficulty
  - ingredients
  - instructions
  - notes
  - rating
- **Meal Plan**
  - ID
  - name of the meal plan
  - meals (for breakfast, brunch, lunch, dinner, snack)
  - notes
- **Ingredient**
  - ID
  - categories
  - quantity
  - unit
- **Schedule**
  - date
  - meal

4.2.2: ER Diagram

Taking into account all the models and requirements, I came up with the following entity-relationship diagram (ER diagram).



It documents the various models, and relations between models. This diagram can be used to derive a schema for the relational database back-end.

4.2.3: Schema

Using the ER diagram above, the following schema was calculated.

Note: underlined fields are primary keys, italic fields are foreign keys and fields where names have a question mark after them are nullable

`meal (`<u>`ID`</u>`, name, time, difficulty, instructions, notes?, rating?)`

`ingredient (`<u>`ID`</u>`, name, quantity, unit)`

`meal_ingredient (`<u>*`meal_ID`*</u>`, `<u>*`ingredient_ID`*</u>`, amount)`

`meal_plan_info (`<u>`ID`</u>`, name, notes?)`

`meal_plan (`<u>`ID`</u>`, `<u>`day`</u>`, `*`breakfast?`*`, `*`lunch?`*`, `*`brunch?`*`, `*`dinner?`*`, `*`snack?`*`)`

`schedule (`<u>`date`</u>`, `<u>`kind`</u>`, `<u>*`meal_ID`*</u>`)`

4.2.4: Routes

With the database itself designed, the routes are also an important consideration as it is an important part of the back-end and how the front end will make requests to the database.

Note there is no authentication in this web application and there isn't any facilities for managing multiple users. For simplicities sake, it can be self-hosted and the instance used by the user / group of users who wish to use it.

All requests and responses will use JSON objects

Meal

- `GET /meals` - gets a list of all meals
- `GET /meal/:id` - gets information about a meal
- `POST /meal` - adds a new meal
- `PUT /meal/:id` - edits information about a meal
- `DELETE /meal/:id` - deletes a meal

Ingredient

- `GET /ingredients` - gets a list of all ingredients
- `GET /ingredient/:id` - gets information about an ingredient
- `POST /ingredient` - adds a new ingredient
- `PUT /ingredient/:id` - edits information about an ingredient
- `DELETE /ingredient/:id` - deletes an ingredient

Meal Plan

- `GET /plans` - gets a list of all meal plans
- `GET /plan/:id` - gets information about a meal plan
- `POST /plan` - adds a new meal plan
- `PUT /plan/:id` - edits information about a meal plan
- `DELETE /plan/:id` - deletes a meal plan

Schedule

- `GET /schedule` - gets a list of all scheduled meals
- `POST /schedule` - adds a new meal to the schedule
- `PUT /schedule/:id` - edits a scheduled meal
- `DELETE /schedule/:id` - deletes a scheduled meal

Chapter 4.3: UI

4.3.1: Theme

Heading 1

Heading 2

Heading 3

Paragraph

#FF1C1C	#FF991C	#FFE51C	#2BFF1C	#1B76FF

Borders: 1px solid

Round corners: 20px

4.3.1.1: Logo

4.3.1.2: Views

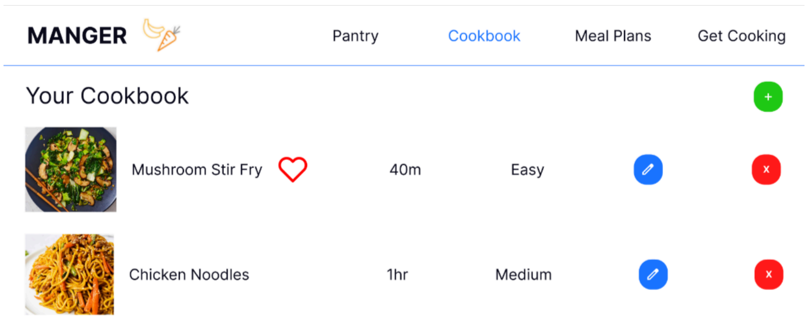
1. **Pantry**
   - Add a new ingredient
   - Remove an ingredient
   - Edit ingredients and their quantities
2. **Cookbook**
   - Add, remove and edit a new recipe
3. **Meal Plans**
   - Create a new meal plan with suggestions when the user looks to add a recipe
   - Edit meal plans
   - Delete meal plans
4. **Get Cooking**
   - View the schedule as it stands
   - Activate meal plans
   - Add meals to the schedule
   - Option to start cooking your meal on the day
   - Cook the meal using meal flow and ask for a rating at the end before going back to the main page

4.3.2: Designs

I created some mock-up designs in figma to show the 4 main views in the application.

These are very rough mock-ups, but outline the general theme and feel of the application and set out the basic layout and idea.

Cookbook



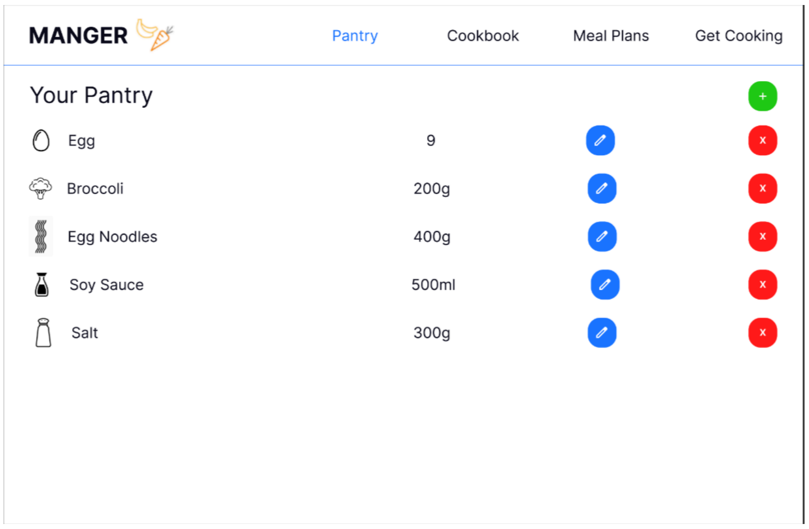
MANGER — Pantry | Cookbook | Meal Plans | Get Cooking

Your Cookbook

	Mushroom Stir Fry	40m	Easy		
	Chicken Noodles	1hr	Medium		

You can add, edit and delete meals from the cookbook. These can then be directly added to the schedule of what you're eating, or added to a meal plan and then added to the schedule.

Pantry



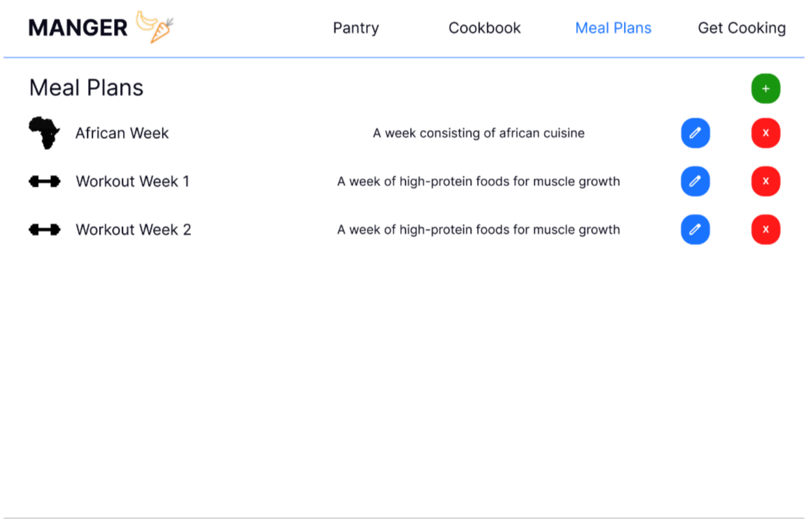
MANGER — Pantry | Cookbook | Meal Plans | Get Cooking

Your Pantry

Egg	9		
Broccoli	200g		
Egg Noodles	400g		
Soy Sauce	500ml		
Salt	300g		

Here you can update your inventory (or pantry as it is called) with what ingredients you already have, and which the software can then use to keep track of what you have, work out what you need to buy, and suggest meals that use existing ingredients up.

Meal Plans

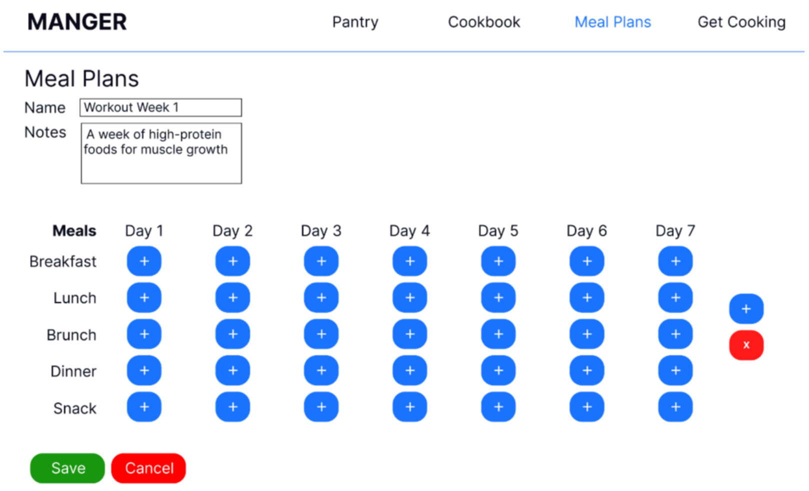


Meal Plans

African Week	A week consisting of african cuisine
Workout Week 1	A week of high-protein foods for muscle growth
Workout Week 2	A week of high-protein foods for muscle growth

Above is the meal planner view, where you can see the list and descriptions of all the meal plans.

From here, you can make new meal plans, and edit or delete existing ones.



Meal Plans

Name: Workout Week 1

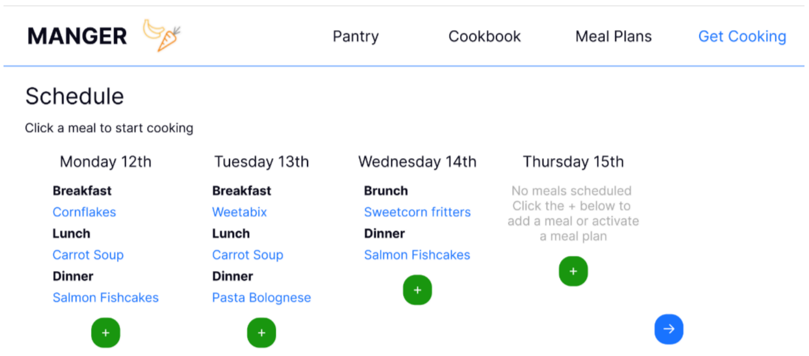
Notes: A week of high-protein foods for muscle growth

Meals	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Breakfast	+	+	+	+	+	+	+
Lunch	+	+	+	+	+	+	+
Brunch	+	+	+	+	+	+	+
Dinner	+	+	+	+	+	+	+
Snack	+	+	+	+	+	+	+

Above is the edit / add page of the meal plan. You give it a name and add some notes if you desire. Then, you can populate the plan with various meals for certain times (like dinner or lunch). Meal plans can consist of a variable number of days. Some people like to shop weekly, others don't mind shopping every 4 days or so.

Once created, you can add them to the schedule starting from any day you wish.

Get Cooking



Schedule

Click a meal to start cooking

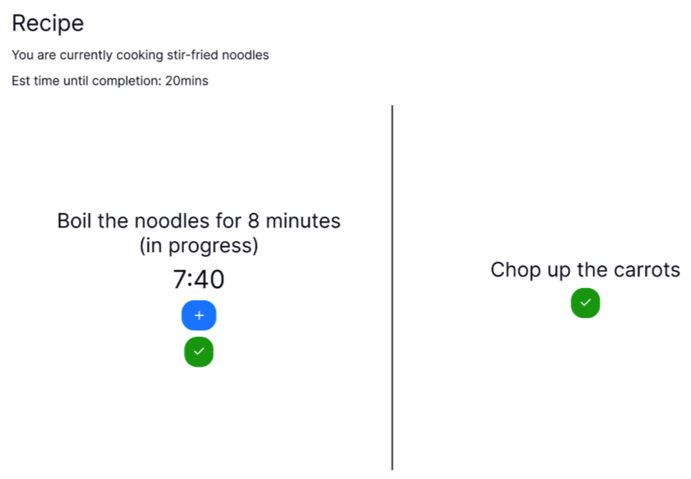
Monday 12th	Tuesday 13th	Wednesday 14th	Thursday 15th
Breakfast	**Breakfast**	**Brunch**	No meals scheduled Click the + below to add a meal or activate a meal plan
Cornflakes	Weetabix	Sweetcorn fritters	
Lunch	**Lunch**	**Dinner**	
Carrot Soup	Carrot Soup	Salmon Fishcakes	
Dinner	**Dinner**		
Salmon Fishcakes	Pasta Bolognese		

Above is the schedule where all scheduled meals are shown. The + buttons are for adding meals to the day, or activating meal plans from that date onwards.

The blue arrow is for seeing further down the line.

It isn't shown here, but there will also be a button to generate a shopping list up to a certain date (e.g. shop for ingredients up to a week from now).

Clicking on the blue links will activate the recipe page, where instructions will be shown.



Recipe

You are currently cooking stir-fried noodles

Est time until completion: 20mins

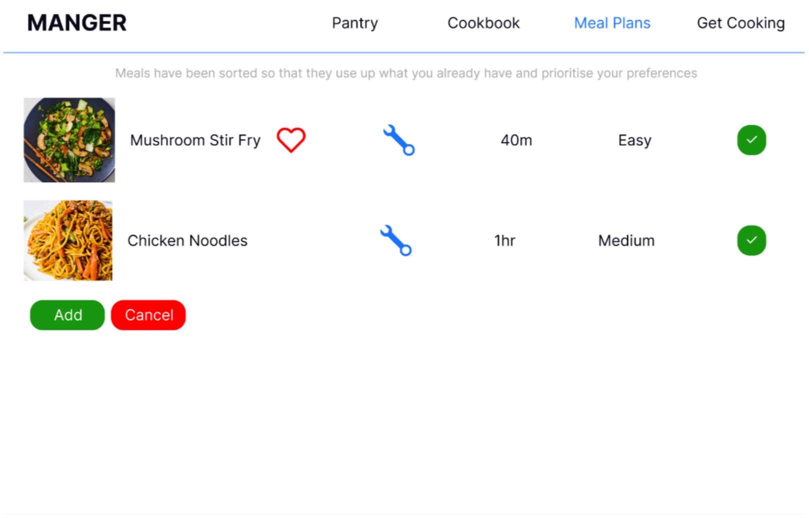
Boil the noodles for 8 minutes
(in progress)

7:40

Chop up the carrots

Above is the recipe page, which implements "meal flow" discussed earlier, with timers and other interactive features. Clicking the check mark means that you have completed that instruction. Pressing the add button on the timer will add a minute to the timer in case you feel it needs more time.

Also shown is the estimated time to completion, which is useful.



MANGER Pantry Cookbook Meal Plans Get Cooking

Meals have been sorted so that they use up what you already have and prioritise your preferences

Mushroom Stir Fry		40m	Easy	✓
Chicken Noodles		1hr	Medium	✓

Above is the meal selection page, where meals are sorted in order of usefulness. Hovering over the wrench will show what ingredients will be used if the meal is to be added to the schedule.

Chapter 4.4: Tools

I will use the PEVN stack

- PostgreSQL
- Express
- Vue
- NodeJS

These are commonly used technologies and will facilitate a seamless client-side single page application.

I will also use libraries like

- Dracula Graph Library (for rendering meal flow diagrams)
- Axios (for linking the front-end to the back-end)
- Vue-router (navigation around the vue application)

Additional technologies may be introduced along the way where needed

Section 5: Workload

I will break down the workload into smaller sub-goals to help build the web application.

1. Back-end
   - Get the database up and running with some randomly generated data
   - Write tests on expected data
   - Set up the API routes for querying data from the database to show to the front-end and ensure the tests pass
2. Pantry functionality
   - Show list of ingredients and their info
   - Add, delete, edit ingredients
   - Write vue tests for the frontend
3. Cookbook functionality
   - Show list of meals and their info
   - Add, delete, edit recipes
   - Write vue tests for the frontend
4. Building meal plans
   - Show list of meal plans
   - Create a new meal plan
   - Meal selection
   - Edit an existing meal plan
   - Delete an existing meal plan
   - Write vue tests for the frontend
5. Meal scheduling
   - Show schedule
   - Allow adding meals to schedule
   - Allow adding meal plans to schedule
   - Allow scrolling on the schedule
   - Write vue tests for the frontend
6. Recipe view
   - Write vue tests for the frontend

Section 6: SDLC

Chapter 6.1: Planning

See objectives, requirements, and user profiles

Chapter 6.2: Analysis

See workload and tools

Chapter 6.3: Design

See design

Chapter 6.4: Testing

For this application, I decided on a test driven development approach. Tests will be written immediately after the boilerplate is up and running. Then the code will be written and modified until tests work.

I will use Vitest as suggested by the VueJS documentation.

Chapter 6.5: Implementation

See workload for a process of implementation. This will be done in stages until the whole application is complete.

Chapter 6.6: Deployment

For the back-end, deployment can be done on AWS, through their RDS service free for 750 hours.

For the front-end, deployment can be done through vercel and hosted forever through the hobby plan, as I don't plan to use this commercially.

Chapter 6.7: Maintenance

As I don't plan to add any additional features in particular, this will likely not be a big stage.