MealsMyWay
Software Testing Plan - Version 1

04/04/2025

Sponsor: Dr. Ana Paula Chaves
Mentor: Paul Deasy

Isaiah Swank
Laura Guerrero
Maximilian Poole
Colin MacDonald

Table of Contents

Introduction

Meal prepping is a valuable way to maintain a healthy lifestyle, save time, and reduce food waste. However, it can quickly become overwhelming due to the effort involved in managing recipes, planning meals, and organizing grocery shopping. For people with busy schedules, meal prepping can help reduce stress and free up time, but the process itself can feel like a lot of work. Additionally, more and more people are looking to add a social aspect to meal prepping. Whether it's sharing recipes, meal plans, and grocery lists with friends or family; which only adds another layer of complexity.

To address these challenges, MealsMyWay provides a flexible, customizable platform available on both web and mobile. Users can fully personalize meal prep calendars, manage recipes, create smart shopping lists based on available ingredients, and collaborate by sharing content with others. As a stretch goal, artificial intelligence may also be used to recommend meals tailored to user preferences and past behavior. To ensure this functionality works as intended, this document outlines our comprehensive software testing strategy. Software testing helps verify that the system meets its requirements and performs reliably. We will conduct unit testing to validate core components like recipe management and calendar logic; integration testing to confirm that modules such as the backend, database, and frontend communicate correctly; and usability testing to evaluate how effectively users can interact with the app.

User experience and convenience are central to MealsMyWay, our testing plan prioritizes usability, followed by integration and unit checks. This approach ensures an intuitive, collaborative, and efficient meal planning experience for all users to allow a more fluent way to create recipes, collaborate with others, and generate grocery/ prep lists.

Unit Testing

Unit testing is the process of verifying that individual components or "units" of code, such as functions, methods, or classes, perform as expected in isolation. The primary goal of unit testing is to detect and fix bugs early in the development cycle by ensuring that each component behaves correctly under various conditions, including typical, boundary, and invalid input scenarios. For the MealsMyWay project, unit testing plays a critical role in validating the core functionality of features like recipe management, calendar creation, shopping list generation, and ingredient tracking. This level of testing ensures that foundational logic remains stable as the application grows in complexity.

To implement this testing approach, we rely on the Jasmine testing framework paired with the Karma test runner. Jasmine provides an expressive and readable syntax for writing behavior-driven test cases, while Karma serves as the environment in which those tests are executed in the browser. Using Angular's HttpClientTestingModule and HttpTestingController, we are able to mock HTTP interactions, simulate various server responses, and ensure that each unit can be tested in true isolation without requiring a live backend. This combination allows us to test business logic, component rendering behavior, and service integration under a wide range of scenarios.

Tests are run using the ng test command, which compiles the test suite and launches a browser showing the Jasmine interface. This interface displays the number of total specs executed, which passed, which failed, and descriptions for each outcome. It is particularly useful for real-time feedback during development, especially when tests are automatically re-run as files are saved and updated.

Jasmine 4.6.1

××•××××••••••••••••••••••••••••••••••••••••••×××•×••

52 specs, 10 failures, randomized with seed 18209

Spec List | Failures

RecipeService
- × should be created

UserService
- × should be created

TabsPage
- • should create

PantryPage
- × should create

CalendarService
- × should be created

Tab3Page
- × should create

SignupPage
- × should create

Tab2Page
- • should not call loadCalendar service if user is missing
- • should clone and add selectedMeal to the calendar in pushMeal()
- • should send calendar invite and show success alert
- • should populate hoveredRecipe and selectedEvent on onRecipeClick()
- • should call saveCalendar with correct payload if user is present
- • should generate 20 weeks of plans including current week
- • should load recipes from navigation state if available

  Utility & Lifecycle Functions
  - • should reload recipes when view is about to enter
  - • should return ingredients from array format
  - • should return parsed ingredients from comma-separated string
  - • should initialize with stored plan and user and call loadCalendar
  - • should format category keys to readable strings
  - • should return null if no numeric value is found
  - • should correctly convert different units to ounces
  - • should parse simple ingredient strings with dash format
  - • should parse old-style format ingredient strings
  - • should save selected plan to sessionStorage and load calendar

- • should call backend and update prep list on generatePrepList()
- • should not call saveCalendar service if user is missing
- • should create
- • should clear hoveredRecipe and selectedEvent when recipe is null
- • should call pushMeal if ingredients and instructions are present
- • should call loadCalendar service if user is present
- • should display grocery list if it exists
- • should alert if grocery list does not exist
- • should alert if prep already exists and calendar hasn't changed
- • should load recipes from sessionStorage if no nav state

  Recipe & Prep List Actions
  - • should remove a recipe and call saveCalendar after confirmation
  - • should display the prep list if available in sessionStorage
  - • should alert if no prep list is available
  - • should clear hoveredRecipe and selectedEvent when closeRecipeDetails is called

- • should alert if meal, day, or category is not selected
- • should set currentWeekStart to the most recent Sunday
- • should initialize currentWeekEvents for a new week
- • should show error alert if calendar invite fails
- • should search users and update searchResults
- • should toggle share calendar view
- • should clear search results on empty query

As seen in the image above, our current test run includes 52 total test specs, with 10 listed as failures. These failures are expected at this stage of development. While we have fully implemented the calendar page test suite, many of the other test files have been scaffolded but not yet filled in. This means the Jasmine test runner attempts to run them, but because they're still empty shells or placeholder test cases, they naturally fail. As we continue building out the logic for pages like Pantry, Profile, Recipes, and Login, these test files will be populated with meaningful, passing test cases.

Our initial testing focus has been on calendar.page.ts, which is one of the most complex components in the app. It manages weekly meal planning, stores and retrieves calendar events, generates grocery and prep lists, and supports collaborative sharing. Unit tests for this page verify the behavior of lifecycle methods (ngOnInit, ionViewWillEnter), ensure that user and recipe data are loaded properly, and confirm that events are added and removed accurately. Utility methods for parsing ingredients, converting units, and formatting data are also covered in depth. Edge cases such as missing users, invalid inputs, and server errors are accounted for to ensure stability in a variety of conditions.

We are now applying the same unit testing strategy across the rest of the app. For pantry.page.ts, test cases will focus on adding, deleting, and updating pantry and freezer items, verifying that user state is loaded before modifying data, and ensuring PantryService methods are called with the correct payloads. The login.page.ts test suite will simulate user authentication with valid and invalid credentials, verify user session persistence, and ensure proper navigation to the calendar page on success. In profile.page.ts, tests will confirm that user settings like email visibility, password changes, and privacy preferences are updated correctly. We will also test the flow of accepting and declining shared calendar invites. Finally, recipes.page.ts tests will validate

recipe form submissions, data fetching from APIs, selection handling, editing, and filtering behavior.

In addition to page-level testing, we are writing full unit tests for backend-integrated service files. For example, CalendarService will be tested to verify all calendar operations such as saving, loading, updating, and sending invites. We will mock HTTP calls and assert correct method execution and payload formatting. Similarly, PantryService tests will confirm pantry and freezer data are posted and updated properly, and ensure that observable updates through triggerPantryReload() function as expected. ProfileService will be tested for privacy and password update logic, as well as shared calendar management. In RecipeService, we will verify that recipes are fetched from the backend or third-party APIs and that data is parsed correctly. UserService tests will validate username management, user persistence in session storage, and user search capabilities.

All test cases follow the Arrange-Act-Assert pattern, and we ensure that each unit is validated against normal, boundary, and error-prone inputs. We use Jasmine spies and mocks to simulate component interactions, isolate dependencies, and simulate backend failures for robustness testing.

To further improve code stability and streamline development, we plan to integrate unit testing into our continuous integration (CI) pipeline using GitHub Actions. Each time a commit is pushed or a pull request is opened, the pipeline will automatically install dependencies, build the project, and run all unit tests in a headless browser using ng test --watch=false --browsers=ChromeHeadless. If any test fails, the CI job will fail, preventing untested or broken code from being merged into the main branch. In the future, we will also integrate code coverage

reports using karma-coverage and enforce minimum thresholds to maintain high-quality standards across the entire codebase.

By combining thorough unit testing across all components and services with automated CI execution, we are creating a development environment that promotes stability, rapid iteration, and confidence in every deployment. This strategy ensures that MealsMyWay continues to scale reliably as new features are added and as more contributors join the project.

Integration Testing

Integration testing is a key component of software development that allows for not only smooth development but also confidence in a working finalized product. Integration testing focuses on the connectivity of the systems components and how data is being transferred between them. This differs from unit testing in a fundamental way. While both aim to verify the success of certain aspects of the system, unit testing focuses on individual functions or components and whether they return expected values, whereas integration testing ensures that data flows correctly and connections between different components work as intended. It is extremely important to ensure that all functions are working as expected, but it is equally important to ensure that those functions are communicating with each other correctly in order to create and maintain a functional development environment and in turn a functional system.

In any given project there are almost always multiple modules and if the information is not correctly passed between these and communication is not occurring as intended the application will not be functional. Web applications are no different and our project falls under that category. There are three major components that need to communicate in our project: Frontend, Backend, and the database. The frontend of our application, or what the user is actually seeing, frequently sends requests to the backend code that in turn accesses data from the database and sends it back to the frontend. All of these modules need to be able to communicate and have multiple routes and functionalities to do so that all need to be verified through integration testing.

The MealsMyWay application utilizes our Ionic frontend to accept user input and send that data to an Express backend that can accept it, process it, and send it back to the frontend. The first step in our integration testing process is to ensure that all of the routes from the backend to the database are functional. Without these routes users cannot even log in to the application and therefore cannot do anything so it is of the utmost importance that all database connections are fully functional at all times. The next step is to ensure that all of the frontend connections to the backend are working as expected. This order of prioritization ensures that the application is first and foremost reachable and then ensures that all functionalities are working properly. Integration testing can take multiple different forms throughout the development process and in our case has been done in different ways and continues to expand as the project goes on.

In the beginning we manually checked these connections by running new code and clicking around to determine what was working and what was not and this worked until the application got larger, then we switched to postman testing during development to ensure our frontend api calls were actually reaching the endpoints that we expected. This was a great tool for quick development testing but was not a scalable solution. We have now started writing tests in our code to ensure that all connections are functional and running after every code change. These tests are required to be run before code updates can be submitted to our repository to ensure that no functionality is lost in the process. We accomplish this by having multiple tests for success, failure, and niche edge case scenarios for all submissions and requests to our database. We are able to view the results from these and compare them to expected outcomes in order to ensure that everything is working properly. As it stands now the tests are specifically to ensure

backend and database connectivity remain intact but frontend to backend communication testing is in the works.

Our current integration testing setup utilizes the Mocha testing framework as well as libraries such as Chai, Supertest, Sinon, and Argon2 to ensure all tests accurately simulate real application functionalities. Mocha provides a clean, readable structure for writing tests by using recognizable keywords such as “describe” and “beforeEach”. These functions help organize test logic into clear, logical blocks, making the tests easy to read and maintain. This structure also makes Mocha especially approachable for beginners who are just getting started with integration testing. Chai was recommended during research into integration testing practices because it works seamlessly with the Mocha framework and offers a readable, expressive way to write assertions within each test block. Its near-English syntax makes it approachable for beginners, helping clearly communicate what each test is checking without complex syntax.

Supertest is a crucial library used in our integration testing as it is used to simulate HTTP requests to our backend without needing to start the server on an actual port. This is what enables our integration testing to actually send and receive information to confirm all of our checks are working properly. Sinon is a crucial tool in our testing setup because it allows us to stub and control the behavior of our PostgreSQL database interactions without needing to access the actual database. This was especially important for testing data submission logic without affecting the live database. By intercepting and simulating responses from the database layer, Sinon enables us to test route logic in isolation while mimicking real-world conditions. Argon2 is the last library utilized in the current integration testing setup as it allows us to hash and verify the

passwords for ensuring the login and signup functionalities are working properly. We plan to use Cypress for frontend-to-backend integration testing, as it provides powerful real-time interaction simulation. This allows us to verify that user actions on the frontend trigger the correct API calls and receive the expected responses from the backend. These tests will cover anything in our code that results in a submission to our backend to ensure that all connectivity between the frontend and backend is working as intended. These integration tests will allow us to future proof this application for our client and allow for possible continuations in development. By creating an application with complete integration testing coverage it ensures that the application will be able to persist without issue for future developers. We already have a firm foundation for testing the backend to database connections and are confident in our approach for ensuring the frontend and backend communications are operating smoothly as well.

Usability/End-user Testing

Usability or End-user Testing is the process of giving the software to outside users to test whether or not they can use it despite their level of familiarity of tech and software as a whole. This is needed as you don't want to submit a piece of software that is so complex that it makes the common person unable to use it correctly or unable to access its features. When programming the software as a team or yourself, it can be very hard to spot possible complex parts as you know the ins and outs of the software and can easily maneuver it which is why it's so necessary to get an outside perspective on it.

To get the testing started, we will go up to people willing and ask if they want to test our product to help with the end product. We will be using our local machines, particularly laptops, and give them the chance to navigate through it themselves without assistance to help gather information from a fresh user that has no experience. We may also go to different parts of campus to get a large varied group of people instead of just people who share our major in tech as not everyone will have that type of background. It can also help us discern whether there is a difference in behavior between someone who has a tech background and those who don't. We will also be asking friends or acquaintances to help in this, to gather a larger group of people to help catch outliers.

After allowing them to fiddle around with the website and letting them freely explore the website we will then guide them through the intended flow and observe their response. Starting at the login page, we will make them make an account or give them a pre-made account if it is deemed as too much of a hassle. We then take them to the calendar page, explain all the buttons

and show how the prep list and grocery list doesn't work because there aren't any added recipes yet. We will show how to change weeks and how to add recipes if they had any selected to add to a given day and explain the reasons we had for it being designed like that. Then, we gather feedback they gave on that page and move onto the next tab which is the recipe tab. We will show the recipes we have by default, show the information that gets displayed when you click on one, and go through the process of how to add it and the formatting of certain parts that are required. Once again, after that is done we gather feedback and move onto another tab which would be the pantry freezer tab. Follow the same theme as the last two, we walk them through adding items to the different sections along with editing and removing them.

We once again gather feedback and quickly go over the profile page and explain how the privacy toggle works along with how to share calendars. After that, we go over the whole system and add recipes successfully to the calendar and generate the prep lists and how that works with the items in the pantry freezer and the selected recipes. We gather feedback for the system as a whole and see if they have any possible improvements or thoughts towards any of the systems or simply one of the tabs without taking the entire project into consideration. It is important to do this as the first blind test will let us know if there's anything unintuitive involved in the layout or construction of the webpage, and the informed walkthrough will teach them about the project and allow them to have some insight into how it works and possibly allow them to think of improvements that we may of missed while making it originally.

Conclusion

The success of a product like MealsMyWay hinges on more than just its features—it depends on reliability, ease of use, and adaptability for a wide range of users. By implementing a layered testing strategy that includes unit testing, integration testing, and usability testing, we are ensuring that every part of the application is thoroughly validated from the inside out. Each type of testing plays a vital role: unit tests confirm that individual components function correctly; integration tests verify communication between frontend, backend, and database layers; and usability tests provide real-world feedback from actual users to refine the experience and address pain points.

This testing strategy has already allowed us to detect and address issues early in development, improve system stability, and ensure that user-facing features work reliably under a wide variety of scenarios. As we continue developing, testing remains an integral part of our workflow—especially with our use of automated testing pipelines to catch regressions before they reach production.

Looking ahead, this foundation will make it easier to add new features, onboard new developers, and adapt to feedback from real users. Our goal is to deliver an application that not only meets technical requirements but also supports the needs of users trying to improve their lives through effective meal prep. With continued attention to testing and feedback, MealsMyWay will be positioned as a smart, sustainable, and collaborative tool for efficient meal planning.