International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 06 June-2026 | Impact Factor: 3.5

Pocketmeal — A Meal Discovery Web App Built on React 19 and Themealdb

[P. Logaiyan, M. Madhan Raj]
Associate Professor, Department of MCA, Sri Malakula Vinayagar Engineering College (Autonomous), Puducherry 605107, India
Email: logaiyan.mca@smvec.ac.in
Post Graduate Student, Department of MCA, Sri Malakula Vinayagar Engineering College (Autonomous), Puducherry 605107, India
Email: maddy08062004@gmail.com

How to Cite this Article:

Raj, M. M. (2026). Pocketmeal — A Meal Discovery Web App Built on React 19 and Themealdb. International Journal of Creative and Open Research in Engineering and Management, 02(6). https://doi.org/10.55041/ijcope.v2i6.146

https://doi.org/10.55041/ijcope.v2i6.146

Abstract

PocketMeal is a meal web application I done together as a learning project to get comfortable with React hooks, the Context API, and working with a third-party REST data source. The app gets its recipe data from The MealDB, which is a free API available online. Users can look through meal categories, open full recipe pages, search for specific meals, and save their favorites — those saved meals stay even after closing the browser. This paper explains how I built the app, the tools I chose, how everything connects, and the bugs I came across. The code is not perfect and some problems are still present as I write this. I felt it was better to explain those issues clearly rather than fix them just before submitting.

1. This Project is designed For

Honestly, I needed a project that used real data. Toy apps with hardcoded arrays only get you so far — at some point you have to deal with async loading states, API responses that are not quite the shape you expected, and state that needs to live somewhere central rather than inside whichever component happened to need it first. A recipe browser seemed like a good fit because the data model is simple enough to understand quickly but has enough going on to make the interesting problems come up naturally.

TheMealDB is free, has decent documentation, and returns a good amount of data per meal — ingredients, measures, instructions, category, region, and a YouTube link if one exists. That meant I could build something that actually felt like a product rather than a demo.

The paper goes through the objectives first, then the tech I picked, then the structure and how each page works, and then a fairly honest section on the bugs I found. Results and conclusion are at the end.

2. Goals for the Project

I have a rough list at the start, partly because the module asked for one, and partly because I had a hope to keep adding features and losing track of what was actually done. The core things I wanted to show were:

- Functional React components with hooks only — no class components anywhere.
- Live API data fetched with Axios, not hardcoded JSON.
- Multiple pages with proper URL routing, so you can bookmark a category or a specific meal.
- A favorites system where the state is accessible from any page, not passed down as props through three layers.
- Favorites that survive a page refresh — meaning localStorage, not just component state.
- A layout that works acceptably on mobile.

I left TypeScript and tests off the list on purpose. In hindsight that was a mistake for a project with this many moving parts — a couple of the bugs I ended up with are exactly the sort of thing a type checker catches in seconds. But I made the call and stuck with it.

3. What I Used and Why

Table 1 shows the full stack. Nothing exotic here — most of it is just the standard React ecosystem in late 2025 versions.

Table 1. Technology stack

Technology	Version	Role in this project
React	19.2.0	UI — all functional components, hooks throughout
rolldown-vite	7.2.5	Build tool and dev server with HMR
React Router DOM	7.13.0	Client-side page routing
Axios	1.13.3	HTTP calls to the meal API
Tailwind CSS	4.1.18	Styling and responsive layout
TheMealDB API	v1 free tier	All recipe data and images

localStorage	Browser Web API	Saving favorites between sessions

The only slightly non-standard choice was rolldown-vite over regular Vite. It is a drop-in swap that uses the Rolldown bundler instead of Rollup, and the cold start on my machine was noticeably faster. It is experimental but I was not doing anything complicated with the build config, so the risk felt acceptable.

Tailwind v4 was whatever came up when I ran npm install. I have used older versions before and the utility-class approach works well when you are putting components together quickly and do not want to name things. No separate stylesheet per component — everything is inline classes.

4. Structure of the App

4.1 No backend

The whole thing runs in the browser. There is no server I wrote — the app bundle gets loaded once, and after that all the data comes from TheMealDB directly over HTTPS. Favorites sit in localStorage. This made deployment simple and kept the scope manageable, though it also means there is no way to sync favorites across devices.

4.2 Folder structure

I split src into sub-folders by concern rather than by feature:

```
src/api/          four functions, one file — all TheMealDB requests
src/components/   Header, MealCard, FavoriteButton, LoadingSpinner
src/contexts/     FavoritesContext — global favorites state
src/hooks/        useFavorites — a standalone hook I ended up not needing
src/pages/        one file per route
```

The hooks folder turned into a source of confusion, which I explain in Section 6.

4.3 Component tree

The app mounts inside a FavoritesProvider so any component can access favorites state. App renders the Header and the route definitions.

```
main.jsx
  └─ FavoritesProvider
       └─ App
            ├─ Header
            └─ Routes
                 ├─ /              Home
```

```
├── /category/:cat      CategoryMeals
├── /meal/:id           MealDetail
├── /favorites          Favorites
└── /search             SearchResult
```

4.4 API calls

src/api/mealdb.js exports four functions and nothing else. Each one is a single Axios call. Page components import whichever function they need and call it inside useEffect.

```
fetchCategories()          →  /categories.php
fetchMealsByCategory(cat)  →  /filter.php?c={cat}
fetchMealById(id)          →  /lookup.php?i={id}
searchMeals(query)         →  /search.php?s={query}
```

Keeping API logic separate from component logic meant I could change the base URL or add an auth header in one place rather than hunting through five files.

5. Page-by-Page Breakdown

5.1 Home page

The home page calls fetch Categories on climb and supply a grid of cards, one per category. Each card is an image and a label, wrapped in a React Router Link pointing to /category/{name}. Loading Spinner shows while the request is pending.

The grid took longer than it should have. I was using grid-cols-4 without setting a gap, and on smaller screens the images were colliding. Took me an embarrassing amount of time to realise that gap-6 needed its own responsive override at the sm breakpoint.

5.2 Category meals

Category Meals reads the category param from the URL with useParams, fires fetchMealsByCategory, and shows the results as MealCards. There is a bug in — the useEffect dependency array is empty instead of containing [category], which means if somehow land on a different category without the component unmounting, you see stale data. It will not cause a visible problem, because the routing installs and re-installs the component every time, but it is technically wrong.

5.3 Meal detail page

This is the most involved page. TheMealDB returns ingredients and their measurements as twenty separate numbered fields — strIngredient1 to strIngredient20, with matching strMeasure fields. I loop through them all and build a list, skipping anything empty. Most meals have six to twelve ingredients so the upper fields are usually blank.

The instructions come back as one big block of text. I output it as a paragraph with the browser handling the wrapping. Splitting on line breaks would look cleaner but the current version is readable.

5.4 Search

Search lives in the Header. The form's submit handler calls useNavigate and pushes /search?q={term}. SearchResult picks up the term with useSearchParams, calls searchMeals, and renders whatever the API sends back. If the query changes, useEffect re-runs because it is in the dependency array — this one actually worked the way it was supposed to from the start.

The no-results state currently renders the string 'Bad' — an obvious leftover placeholder. I noticed it during testing and never went back to fix it before submission.

5.5 Favorites

Any MealCard can be favorited using the heart button. FavoritesContext holds the array, a toggle function, and an isFavorite predicate. On every state change the context writes to localStorage via useEffect. On startup it reads from localStorage to restore the previous session.

The toggle logic is straightforward — if the meal is already in the array remove it, otherwise append it. MealCard calls isFavorite to decide which icon to show.

6. Bugs and Mistakes

I want to document these properly rather than skipping over them. Some are fixed, most are not.

6.1 FavoritesProvider does not render its children

This one is embarrassing. In FavoritesContext.jsx the component is defined like this:

```
export const FavoritesProvider = ({ Children }) => {
```

Capital C on Children. React passes child elements through the lowercase children prop. The capital-C version is always undefined, so the JSX inside the provider renders is nothing — the entire app below the provider just disappears silently. The fix is obvious in review: lowercase the C. But because React does not throw an error when you reference an undefined variable inside JSX, this fails silently and is not easy to spot just by looking at the screen.

6.2 Hook referenced instead of called

In MealDetail.jsx:

```
const {toggle, isFavorite} = useFavorites;
```

No parentheses. That line is destructuring the function object itself, not the return value of the hook. So toggle and isFavorite are both undefined. The same problem is in Favorites.jsx. Both should be useFavorites(). I caught this one during manual testing when clicking the heart button did nothing.

6.3 Wrong variable in the favorites empty check

The Favorites page has:

```
if (Favorites.length === 0)
```

Favorites with a capital F is the component function, not the state array. Function.length returns the number of parameters, which here is 0 since the component takes no props. So Favorites.length is always 0, meaning the empty-state message always shows — even when there are saved meals. It should be favorites.length from the hook. I spotted this late and still have not fixed it.

6.4 Two implementations doing the same job

There is a useFavorites hook in src/hooks/ and another inside FavoritesContext.jsx. Both manage the same localStorage key independently. MealCard imports from the hook file, MealDetail was supposed to use the context — so in theory they could have different views of the favorites array. In practice the bugs elsewhere mean neither fully works, but if they did, this would cause a subtle sync bug. The right approach is one implementation in the context with the hook re-exported from there.

6.5 CategoryMeals stale data risk

useEffect in CategoryMeals has [] as its dependency array. The category param from the URL is never listed. This is fine as long as React install and re-installs the component on every navigation, which it currently does. But it is incorrect code and would break if the routing ever changed to keep the component alive and just update the param.

7. How It Turned Out

The bugs in the favorites feature mean that part of the app is fairly broken. But browsing categories, clicking into meals, and searching all work fine — those paths do not touch the broken favorites plumbing. A user who ignores the heart button gets a functional experience.

The responsive layout is decent. Category grid drops to two columns on small screens, the meal detail page stacks vertically below about 640px, and the search results grid handles narrow viewports reasonably. I tested down to 375px and nothing overflows badly.

What I learned from the bugs was probably more valuable than what I learned from the things that worked. The capital-C children mistake taught me to be more careful about React's case sensitivity rules. The missing parentheses on the hook call was a reminder that JavaScript will let you reference a function without calling it and will not complain. And the duplicate state problem reinforced why having one source of truth matters — it sounds like a principle from a textbook until you actually break something because of it.

8. Where to Go From Here

If I kept working on this the priority list would be: fix the capital-C children prop, add parentheses to both useFavorites calls, correct the favorites empty-state variable, delete the standalone hook, and put 'category' in the CategoryMeals dependency array. The search empty-state placeholder also needs a real message. Those are all small changes that would make the app genuinely functional.

The bigger improvement would be TypeScript. I mentioned it in Section 2 as something I deliberately left out, and I stand by that decision for a first project — but the capital-C bug and the missing-parentheses bug would both have been caught at compile time with even basic type annotations. The cost-benefit makes a lot more sense on a second pass.

Deployment is trivial since there is no server — the built output is just static files. Vercel or Netlify would handle it with a single command. The only thing worth checking before going public is whether the TheMealDB free tier has any CORS restrictions or rate limits that would affect real-world traffic.

References

React Team. (2024). React Docs. https://react.dev

[2] Vite contributors. (2024). Vite — next gen frontend tooling. https://vitejs.dev

[3] Remix Software. (2024). React Router v7. https://reactrouter.com

[4] TheMealDB. (2024). API documentation for TheMealDB. https://www.themealdb.com/api.php

[5] Axios contributors. (2024). Axios HTTP client. https://axios-http.com

[6] Tailwind Labs. 2023. Tailwind CSS v4. https://tailwindcss.com