

# Raycast Extension Project Structure


TypeScript and React-based Raycast extension with commands, views, and preferences.

Updated 2025-12-22

#raycast #extension #macos #typescript #react

 PNG

 PDF

 Copy

 Prompt

## Project Directory

### my-extension/

- package.json Extension manifi...
- src/ Source code
  - search.tsx Search command
  - create.tsx Form command
  - menu-bar.tsx Menu bar command
- components/ Reusable React ...
  - ListItem.tsx
  - EmptyView.tsx
- hooks/ Custom React ho...
  - useData.ts Data fetching h...
  - usePreferences.ts
- lib/ Utilities and A...
  - api.ts External API ca...
  - utils.ts
  - types.ts
- assets/ Icons and images
  - icon.png Extension icon ...
  - command-icon.png
- tsconfig.json
- .gitignore
- README.md

## Why This Structure?

Raycast uses React for UI with built-in components like `List`, `Form`, `Detail`, and `Action`. Each command is a separate file in `src/` that exports a React component. The `package.json` defines commands, preferences, and metadata.

## Key Directories

**src/\*.tsx** - Each file is a command entry point

**src/components/** - Shared React components

**src/hooks/** - Custom hooks for data and state

**assets/** - Extension and command icons

## Basic Command

```
// src/search.tsx
import { List, ActionPanel, Action } from "@raycast/api";

export default function Command() {
  return (
    <List>
      <List.Item
        title="Hello World"
        actions={
          <ActionPanel>
            <Action.CopyToClipboard content="Hello!" />
          </ActionPanel>
        }
      />
    </List>
  );
}
```

## Getting Started

1. Install Raycast from raycast.com
2. `npx create-raycast-extension` to scaffold
3. `npm install` to install dependencies
4. `npm run dev` to start development mode
5. Extension appears in Raycast automatically

## Core Components

**List** - Searchable list with sections and items

**Form** - Input forms with validation

**Detail** - Markdown content with metadata

**ActionPanel** - Context menu with actions

## Best Practices

- Use `useCachedPromise` for data fetching with caching
- Store sensitive data with `LocalStorage` API
- Define preferences in `package.json` for user settings
- Use `showToast` for feedback on actions
- Keep commands fast—users expect instant response

## Package Configuration

The `package.json` contains Raycast-specific fields: `commands` array defines each command with name, title, and mode. `preferences` defines user-configurable settings. Icons are referenced from the `assets/` folder.

## When To Use This

- Quick access to external services
- Productivity tools and workflows
- Developer utilities and lookups
- Clipboard managers and snippets
- API integrations with visual UI

## Trade-offs

**macOS only** - Raycast is exclusive to macOS

**Store review** - Public extensions require Raycast review

**React required** - Must use React, no plain JS option