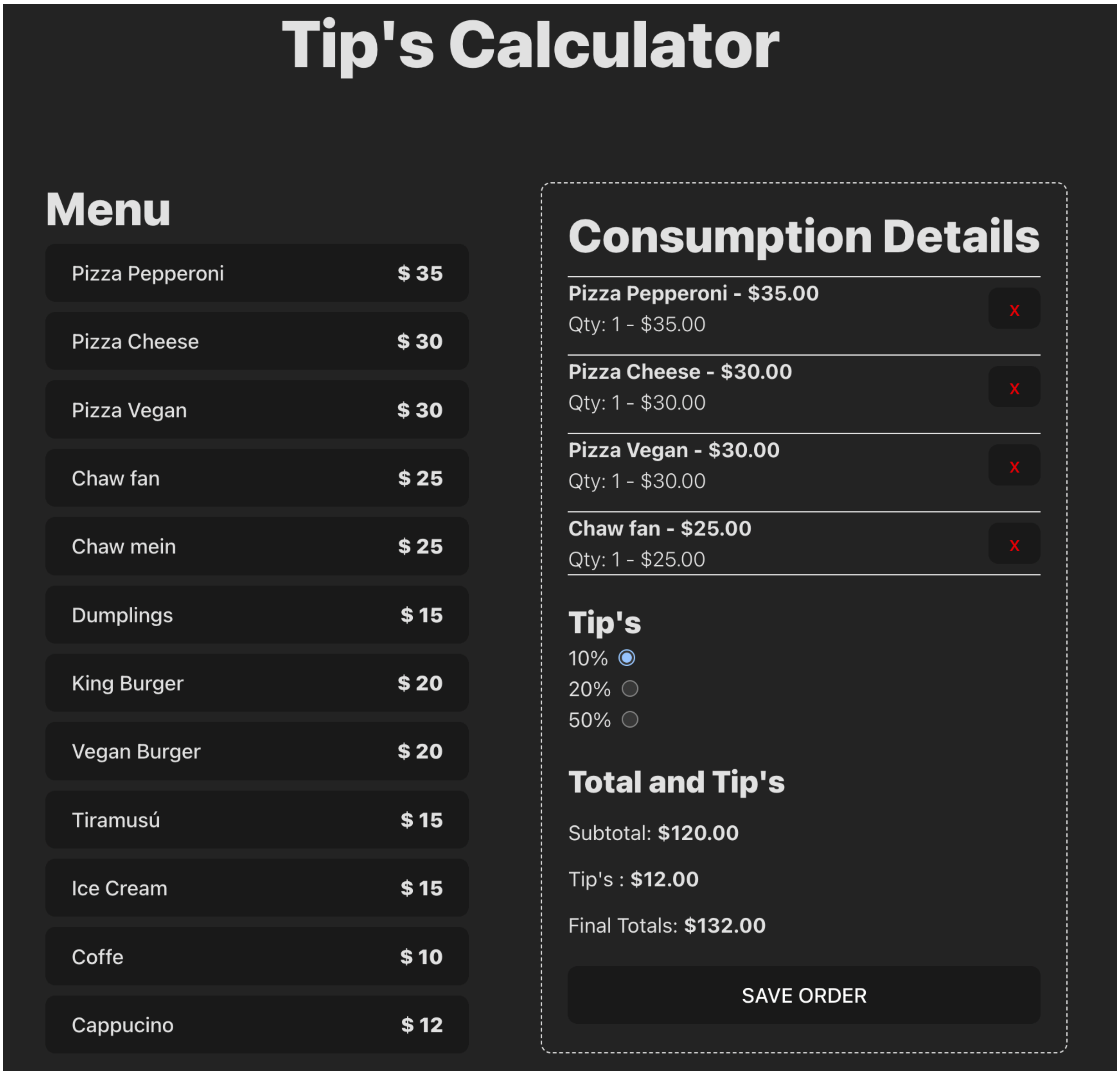
Tip's Calculator

A modern React + TypeScript web application for calculating tips and managing orders



Menu

Pizza Pepperoni	$ 35
Pizza Cheese	$ 30
Pizza Vegan	$ 30
Chaw fan	$ 25
Chaw mein	$ 25
Dumplings	$ 15
King Burger	$ 20
Vegan Burger	$ 20
Tiramusú	$ 15
Ice Cream	$ 15
Coffe	$ 10
Cappucino	$ 12

Consumption Details

Pizza Pepperoni - $35.00
Qty: 1 - $35.00 x

Pizza Cheese - $30.00
Qty: 1 - $30.00 x

Pizza Vegan - $30.00
Qty: 1 - $30.00 x

Chaw fan - $25.00
Qty: 1 - $25.00 x

Tip's

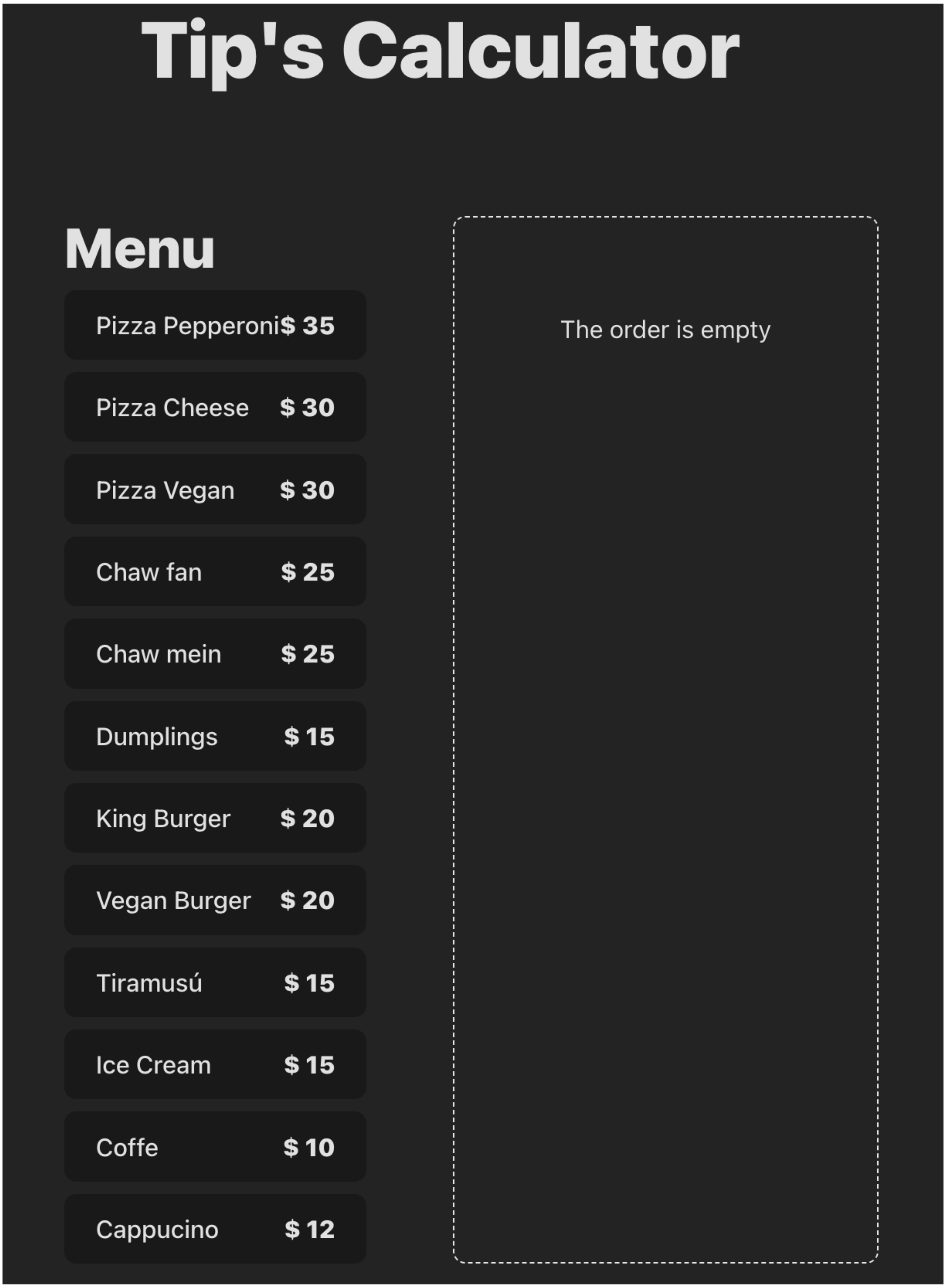
[✓ 10%]
20%
50%

Total and Tip's

Subtotal: **$120.00**

Tip's : **$12.00**

Final Totals: **$132.00**



Tip's Calculator

Menu

Pizza Pepperoni	$ 35
Pizza Cheese	$ 30
Pizza Vegan	$ 30
Chaw fan	$ 25
Chaw mein	$ 25
Dumplings	$ 15
King Burger	$ 20
Vegan Burger	$ 20
Tiramusú	$ 15
Ice Cream	$ 15
Coffe	$ 10
Cappucino	$ 12

The order is empty

About The Project

Tip Calculator is an interactive web application built with **React**, **TypeScript**, and **Tailwind CSS** that simplifies tip calculations and restaurant order management. This project showcases modern React development practices including custom hooks and component-based architecture.

Learning Journey

This is a **learning project** created to master React fundamentals and modern frontend development. As such, it focuses on core React concepts and will be expanded with additional features over time.

Current Status:

- Fully functional frontend with React + TypeScript
- Custom hooks for state management
- Performance optimizations implemented
- Backend integration planned for future versions

Project Highlights:

- Built to master React fundamentals and advanced hooks
- Optimized rendering with `useMemo` and `useCallback`
- Styled with Tailwind CSS utility classes
- Custom hook architecture for reusable logic
- Type-safe development with TypeScript

Live Demo

(add link)

The application provides an intuitive interface for:

1. **Selecting menu items** with instant order updates
2. **Calculating tips** with multiple percentage options
3. **Managing orders** with add/remove functionality
4. **Viewing real-time calculations** of subtotals and totals

Features

Interactive Menu

- Click-to-add menu items
- Dynamic quantity management
- Real-time price updates
- Organized menu categories

Smart Calculations

- Automatic subtotal computation
- Multiple tip percentage options
- Instant total calculations

- Real-time order updates

Order Management

- Add items with single click
- Remove items individually
- Quantity tracking per item
- Empty state handling

Performance Optimized

- Custom hooks for logic reuse
- Memoized calculations
- Optimized re-renders
- Efficient state management

Project Architecture

Component Structure

```
src/
├── components/
│   ├── MenuItem.tsx          # Individual menu item button
│   ├── OrderContents.tsx     # List of ordered items
│   ├── OrderTotals.tsx       # Subtotal, tip, and total display
│   └── TipPercentage.tsx     # Tip percentage selector
├── hooks/
│   └── useOrder.ts           # Custom hook for order management
├── types/
│   └── index.ts              # TypeScript type definitions
├── data/
│   └── db.ts                 # Menu items data
└── App.tsx                   # Main application component
```

Performance Optimizations

Understanding React Hooks for Performance

This project implements several performance optimization techniques:

Custom Hooks

What are Custom Hooks? Custom hooks are reusable functions that encapsulate stateful logic. They allow you to extract component logic into reusable functions.

Benefits:

- Code reusability across components
- Separation of concerns
- Easier testing
- Cleaner component code

In this project: The `useOrder` hook centralizes all order management logic, making the `App` component cleaner and the logic reusable.

```
// Instead of managing state in App.tsx
const [order, setOrder] = useState([])
const [tip, setTip] = useState(0)
// ... all the logic

// We use a custom hook
const { order, addItem, removeItem, tip, setTip, placeOrder } = useOrder()
```

Key Learnings

This project was an incredible journey into React and modern frontend development. Here's what I learned:

React Fundamentals

- **Functional Components**: Modern React development using functions instead of classes
- **Component Composition**: Breaking UI into reusable, manageable pieces
- **Props Flow**: Passing data down from parent to child components
- **Conditional Rendering**: Showing different UI based on state
  ```
  // Example: Conditional rendering in App.tsx
  {order.length > 0 ? (
  <OrderContents order={order} removeItem={removeItem} />
  ) : (
  <p className="text-center mt-10">The order is empty</p>
  )}
  ```
- **useState**: Managing component state
- **useCallback**: Optimizing function references
- **useMemo**: Optimizing expensive calculations
- **Custom Hooks**: Creating reusable stateful logic
- **Interface Definitions**: Creating type-safe data structures
- **Type Annotations**: Ensuring function parameters are correctly typed
- **Generic Types**: Using TypeScript generics with React
- **Type Safety**: Catching errors during development
- **Memoization Strategies**: When and how to use `useMemo` and `useCallback`
- **Render Optimization**: Preventing unnecessary component re-renders

- **Component Splitting**: Breaking down components for better performance
- **State Management**: Efficient state updates with immutability
- **Custom Styling**: Rapid prototyping without writing CSS

Optimization techniques applied:

1. Split `OrderContents`, `OrderTotals`, and `TipPercentage` into separate components
2. Used `useCallback` for event handlers passed as props
3. Applied `useMemo` for calculating subtotals and totals
4. Implemented efficient array operations (map, filter, reduce)

Current Limitations & Future Improvements

As this is a learning project focused on mastering React fundamentals, some features are intentionally simplified or not yet implemented:

Current State

What's Working:

- Complete frontend functionality
- Order management (add, remove, calculate)
- Real-time calculations
- Responsive UI with Tailwind CSS
- Type-safe code with TypeScript
- Performance optimizations

⏳ **What's Next:**

- **No Data Persistence**: Orders are not saved to a database
  - The "SAVE ORDER" button currently only logs to console: `console.log('Order Saved Successfully!')`
  - After clicking, the order resets to allow for a new order
  - This is a placeholder for future backend integration
- **No Backend Integration**: Currently a frontend-only application
  - Will integrate with a REST API or GraphQL backend
  - Planning to use Node.js + Express or similar
- **No Order History**: Previous orders aren't stored or retrievable
  - Will implement order history with database storage
- **No User Authentication**: No login/signup functionality
  - Will add authentication in future versions

Why These Limitations?

This project was built to focus on and master:

1. **React fundamentals** (components, hooks, state management)
2. **TypeScript integration** (types, interfaces, type safety)
3. **Performance optimization** (useMemo, useCallback, custom hooks)
4. **Modern styling** (Tailwind CSS)

Backend development and data persistence are planned for **Phase 2** of this project once the frontend foundations are solid.

📝 License

Author

Micaela Videla Melo

- GitHub: GibHub
- Linkedin: Linkedin

Contact

Have questions or suggestions? Feel free to reach out!

- Email: micaelavidelamelo@gmail.com