


# Michael Trapani

Software Engineer

(347) 886-4875 

michaeltrapani.mail@gmail.com 

michaeltrapani.io 

linkedin.com/in/michael-a-trapani 

github.com/michaeltraps 

New York, NY 

## SKILLS & TECHNOLOGIES

**Languages:** JavaScript (ES6+), HTML, CSS/SASS, SQL, GraphQL

**Libraries and Frameworks:**

React, Redux, Material-UI, Node.js, Express.js, Mongoose, Jest, TypeScript

**Tools:** Git, Webpack, Docker, AWS, CI/CD

**DBMS:** MongoDB, PostgreSQL

## EDUCATION

**Codesmith**

Advanced Software Residency Program

**Stevens Institute of Technology**

ME, BE Biomedical Engineering  
Cert. in Project Management

## TALKS & PUBLICATIONS

**Popular Frontend Frameworks: A Discussion on Angular, Vue, and Svelte**

Single Sprout

**SQuriL: Generate and Store Your GraphQL Schema**

Medium.com

## INTERESTS

**Running** in local races

**Traveling** to hiking destinations, both international and domestic

**Reading** classic and contemporary science fiction

**Playing** RPG, strategy, simulation games

## EXPERIENCE

**SQuriL** | GraphQL schema generation tool for PostgreSQL databases

2022

Frontend Software Engineer | Open Source Labs

- Developed **GraphQL** schema generation tool that creates production-ready GraphQL schemas with TypeScript typing from a PostgreSQL database, improving dev velocity
- Used **React** to leverage the virtual DOM's speedy reconciliation process coupled with reusable and accessible components handling large state updates, necessary for seamless display of complex GraphQL schemas for optimal user experience
- Leveraged **Material-UI** to present an accessible interface and facilitate ease of integration with existing React components, implementing theme-setting functionality to create palettes, as well as developed light and dark mode styling
- Architected **Express** backend for its customizable and chainable middleware capabilities to read and process **PostgreSQL** databases and return GraphQL schemas for use with Apollo server to display on the application frontend
- Integrated support for **TypeScript** codebases via the robust plugin library available with GraphQL Code Generator by automatically populating GraphQL schema output with base TypeScript types consistent with database schema structure
- Utilized the compatibility of **Jest** and **React Testing Library** to create a testing suite for 100% of all key application components, ensuring a reliable engineering environment, higher quality of code, and easier future feature integration

## OPEN SOURCE

**GitGood** | Learning tool that stores online resources in customizable categories

- Built a lightweight SPA with **React** that allows users to create, update, and display customizable topic and subtopic study flashcards, leveraging component state management via hooks to generate a seamless user interface
- Applied **SASS** to create consistently-themed and accessible styling for frontend dashboard display and flash cards, paired with a robust component and icon library to produce a visually appealing, clean and readable study interface
- Architected an **Express** server utilizing customized middleware, routing, and error handling to manage a scalable volume of users to handle different HTTP requests to endpoints for topic and subtopic study flashcard editing

**NutritionX** | Nutrition tracker that allows users to track their meal nutrition info

- Utilized **React** to manage tech debt by refactoring legacy components to accelerate dev velocity and quickly and dynamically display nutritional information fetched from an external nutrition API
- Appended and refactored an **Express** server with additional routes and endpoints that allow users to save their favorite meals, well as display filtered saved meals to a redesigned and reformatted frontend display for enhanced user experience

**GoodWatch** | Movie library platform for leaving ratings and reviews, and saving movies

- Designed frontend codebase using **React** and **Redux Toolkit** to leverage built-in Redux Thunk functionality to perform asynchronous data fetches via GraphQL queries and dispatch async actions from a single, easily readable location