Supermarket Path Planner

Project number: 15006014

Workshop: Cloud computing

Team members: Guy Sofer, Ben Hanover, Alicia Belhassen

Workshop director: Yogev Shani

Background & Problem

- Grocery shopping in large stores takes a lot of time. Signs are often unclear, and layouts can change, making it hard to find items without going back and forth.
- Unlike outside, where we use GPS to get around, there's nothing like that inside stores.
- The Supermarket Path Planner helps by giving shoppers a mobile app that shows the shortest route based on their grocery list.

Our solution

We built a cloud-powered system with two parts:

1. A web platform - for store owners to manage store layout and products

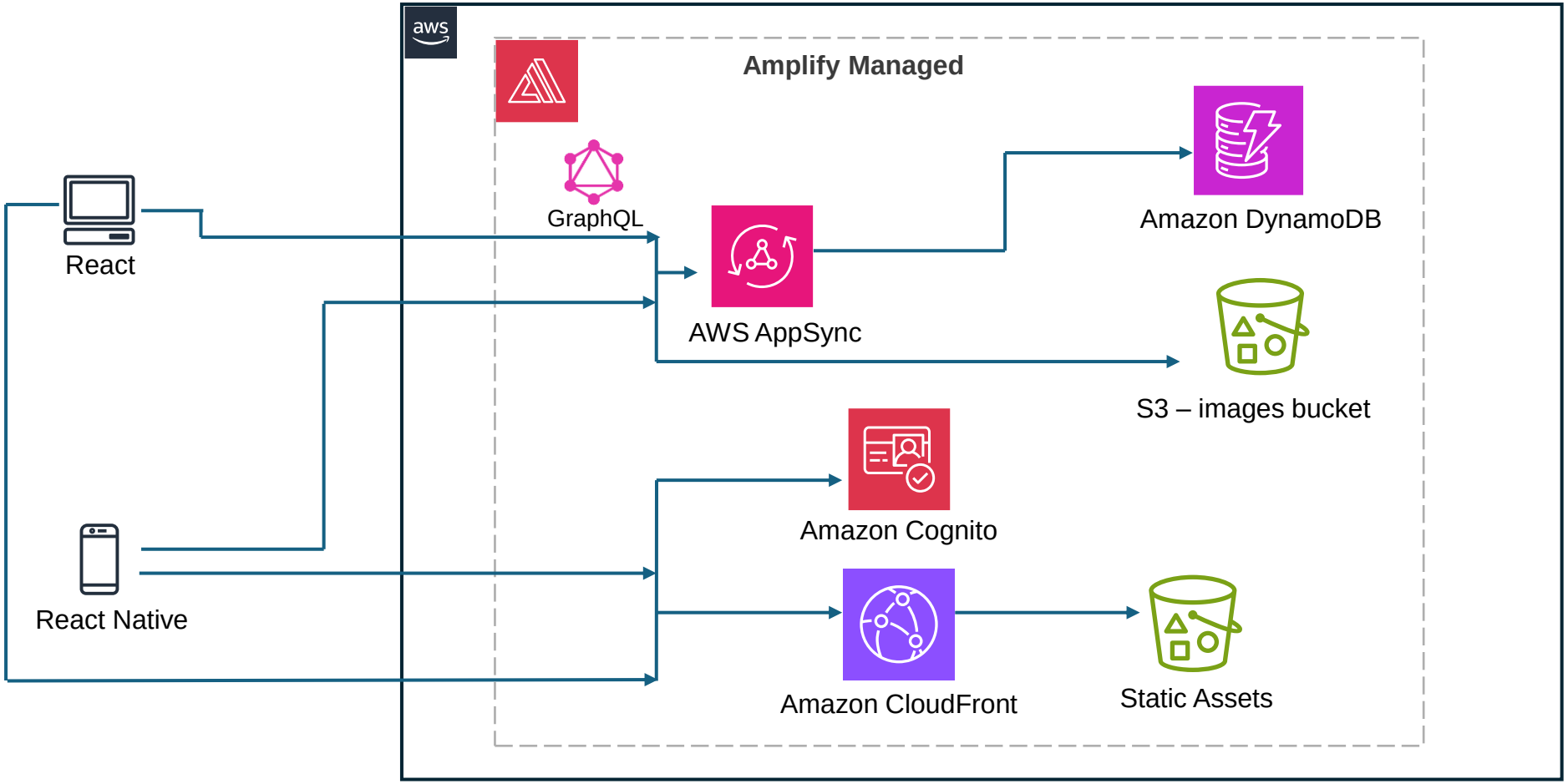
2. A mobile app - that gives customers the shortest shopping route based on their grocery list.

This platform is fully responsive and cloud-integrated using AWS Amplify, AppSync, DynamoDB, Cognito, and S3.

Web Platform - Dashboard Demo

Mobile app Demo

AWS Cloud Architecture



Shortest Route Construction

Building the Graph

The graph construction transforms the 2D supermarket layout into a weighted adjacency matrix by converting each grid square into a node and connecting walkable neighbors with appropriate edge weights.

Floyd-Warshall Algorithm

computes all-pairs shortest paths using dynamic programming.
The algorithm produces distance and next-hop matrices that enable O(1) path queries after one-time preprocessing, with results cached in DynamoDB for instant route planning.

TSP Using Held-Karp Dynamic Programming

The Held-Karp algorithm solves TSP optimally using bitmask dynamic programming with $O(n^2 \times 2^n)$ complexity, where n represents the number of products to collect. For shopping lists exceeding 15 items, the system automatically switches to a nearest-neighbor heuristic with $O(n^2)$ complexity to maintain sub-second response times.

Route Reconstruction

Route reconstruction uses BFS pathfinding with $O(n \times (V + E))$ complexity to generate step-by-step walking directions between TSP waypoints, ensuring all paths traverse only walkable squares.

Other solutions **VS.** Our solution

Most grocery apps focus on online ordering or provide static store maps, which don’t help much with in-store navigation.

While some offer basic tools like inventory lookup, they often don’t consider the shopper’s specific needs.

This project stands out by using smart pathfinding algorithms tailored to each customer’s shopping list, along with a platform for store owners to manage store layouts.

Q&A

Database Schemas

Product	
id	ID (Primary Key)
title	String
price	Float
category	String
image	String (S3 Path)
supermarketID	Foreign Key

Supermarket	
id	ID (Primary Key)
owner	String (Required)
name	String
address	String
layout	JSON (Grid Data)
pathData	JSON (Floyd-Warshall)

ShoppingList	
id	ID (Primary Key)
name	String
owner	String
productIDs	JSON Array
createdAt	DateTime
completedAt	DateTime

Algorithm Implementation

1. Building the Graph

The graph construction phase transforms our 2D supermarket layout into a weighted adjacency matrix suitable for pathfinding algorithms. Each square in the grid becomes a node, with indices calculated using the formula row × cols + col. The algorithm iterates through every square in O(rows × cols) time, examining up to 8 neighbors (4 orthogonal and 4 diagonal directions) for each position. Walkable squares (empty spaces) receive standard edge weights of 1.0 for orthogonal movements and $\sqrt{2}$ (≈1.414) for diagonal movements, while product squares allow entry/exit with a slight penalty weight of 1.1 to discourage unnecessary traversal. Non-walkable obstacles receive infinite weight, effectively blocking those paths. The resulting adjacency matrix requires $O(V^2)$ space where V represents the total number of squares, creating a comprehensive graph representation that enables efficient shortest-path computations across the entire store layout.

2. Floyd-Warshall Algorithm

The Floyd-Warshall algorithm computes all-pairs shortest paths using dynamic programming with a time complexity of $O(V^3)$, where V equals the total number of grid squares. This triple-nested loop structure systematically considers each vertex k as an intermediate point, testing whether the path from vertex i to vertex j through k yields a shorter distance than the current direct path. The algorithm maintains two matrices: a distance matrix storing the shortest path lengths and a next-hop matrix enabling path reconstruction.

The algorithm's results are serialized as JSON and cached in DynamoDB, ensuring that subsequent route optimizations can leverage precomputed shortest paths without recalculation, making real-time customer route planning feasible.

3. TSP Using Held-Karp Dynamic Programming

The Traveling Salesman Problem (TSP) optimization employs the Held-Karp algorithm, a dynamic programming solution with $O(n^2 \times 2^n)$ time complexity and $O(n \times 2^n)$ space complexity, where n represents the number of product locations to visit. The algorithm uses bitmask representation to track visited cities and maintains a state table dp[mask][i] representing the minimum cost to visit all cities in the bitmask ending at city i. For each state, it explores all possible next destinations, updating the optimal path when a shorter route is discovered. While this exponential complexity limits practical usage to approximately 15-20 products, it guarantees mathematically optimal solutions for smaller shopping lists. For larger lists, the system automatically falls back to a nearest-neighbor heuristic with $O(n^2)$ complexity, providing 90-95% optimal solutions in under 50 milliseconds. The algorithm considers the entrance as the starting point and outputs an ordered sequence of product access points that minimizes total walking distance while respecting store layout constraints.

4. Route Reconstruction

Route reconstruction transforms the abstract TSP solution into concrete, step-by-step walking directions through a multi-stage process with $O(n \times (V + E))$ total complexity, where n is the number of products and V, E represent vertices and edges in the walkable area. The system first identifies walkable access points adjacent to each product square, then uses breadth-first search (BFS) with $O(V + E)$ complexity to generate collision-free paths between consecutive waypoints in the TSP sequence. Each BFS operation explores only empty squares, ensuring customers never walk through product displays or blocked areas. The algorithm processes four distinct route segments: entrance to first product access point, sequential paths between product access points following TSP order, navigation to checkout/cash register area, and finally the path to store exit. Path validation removes duplicate consecutive coordinates and verifies walkability of every step, while the mobile application receives numbered waypoints with associated product lists for each stop.