OptiRide

Optimized Ride Scheduler

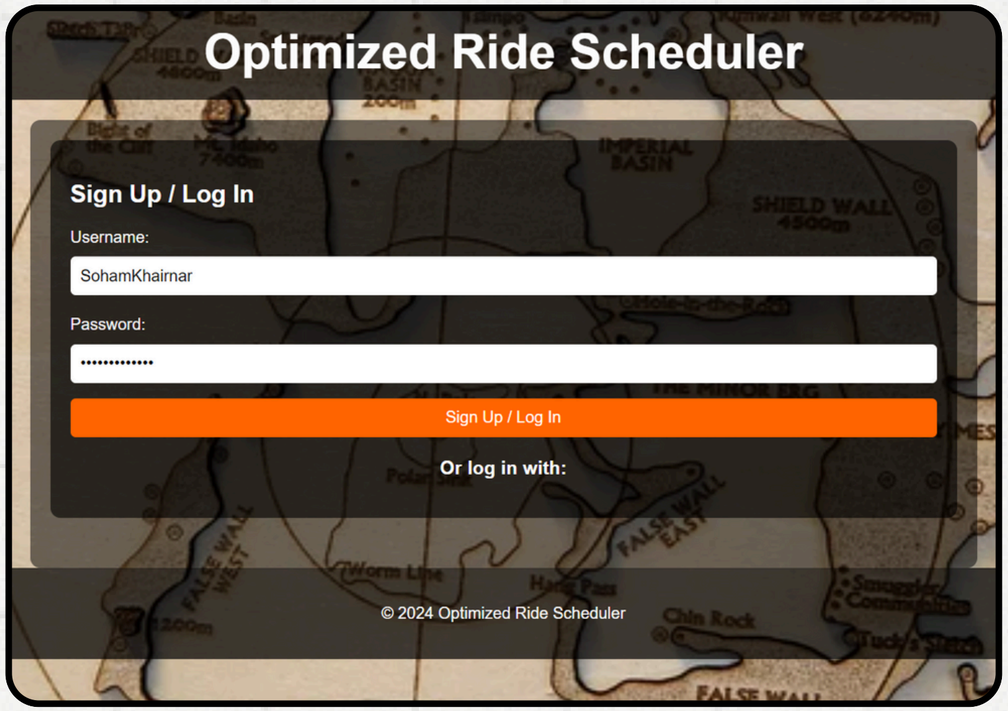
By Soham Khairnar (B23CM1039 , Ishan Shah (B23CM1050) , Vaibhav Garg (B23CM1046) & Nishchal Badaya (B23CM1053)

About OptiRide

- **Objective :** Design an algorithm for real-time ride-scheduling that efficiently matches passengers with drivers by minimizing waiting times and travel distances, while dynamically adapting to new requests and driver status updates. The project may also explore ride-pooling to optimize shared routes.

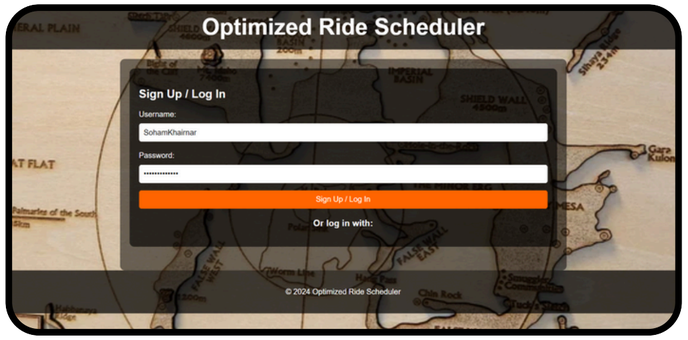
- **Main Task :** Schedule rides from one point to another accounting the intermediate locations and minimizing the cost of transport

- **Tech Stack :** C++ , Python, HTML, CSS and Javascript



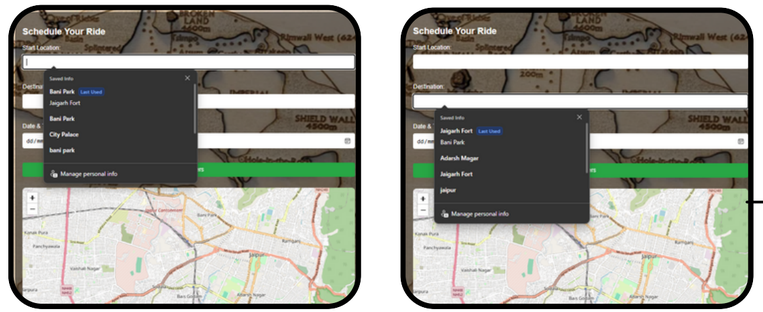
Working

Welcome to OptiRide



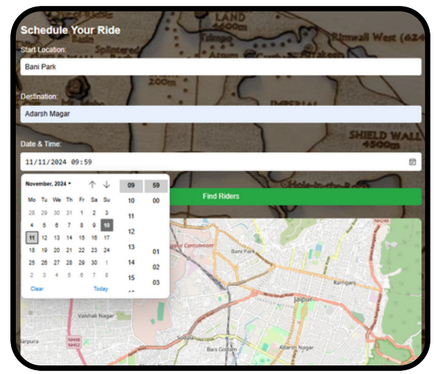
Open OptiRide and sign up / login

Set the Locations



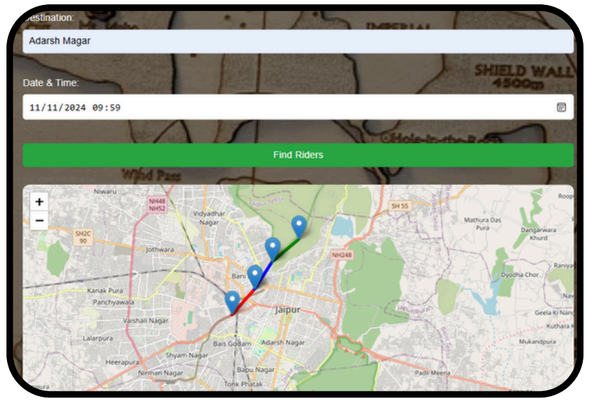
Set the locations after signing in / log in

Set Date and Time



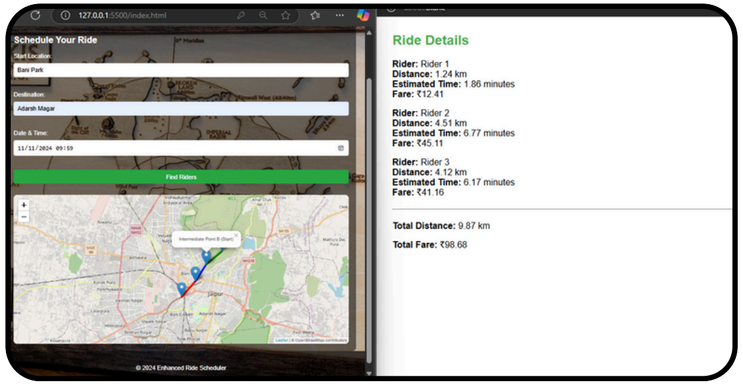
Set the date and time of the ride

See the Shortest Route



You will see the shortest route

Get Ride Details



Get the ride details with distance & price

CODE

A* SEARCH

This Algorithm runs the code. It is used for its efficiency, heuristic guidance, optimality and flexibility.

STACK

A stack allows LIFO (Last In, First Out) traversal, which is efficient for reversing the path from destination to source.

DOM MODEL

Used to create the priority backend of the site.

ARRAYS

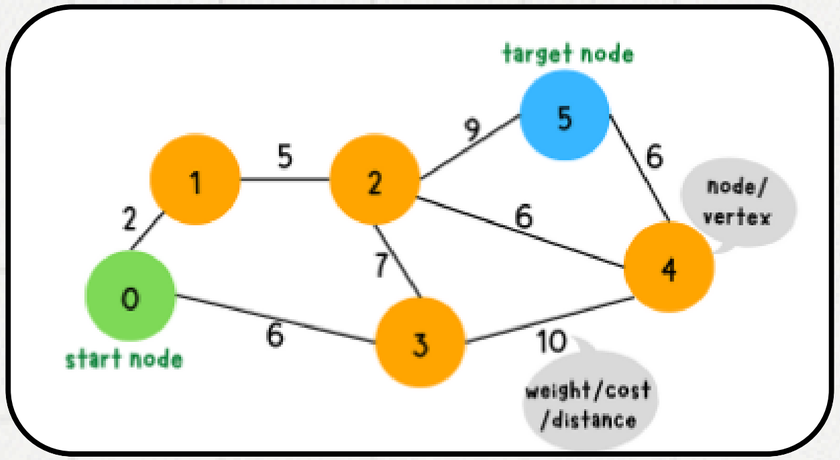
Vectors have been used for creating bools, grids, storing coordinates etc.

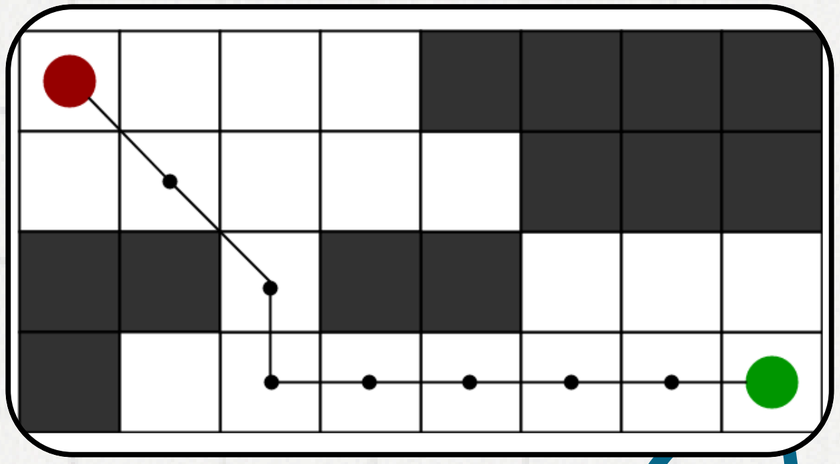
MIN. HEAP

The min-heap property allows retrieving the cell with the minimum f cost in O(logn time

A* Algorithm

- This project implements the A* search algorithm in C++ to find the shortest path from a source to a destination on a 2D grid. A* algorithm is a popular choice for pathfinding as it combines both path cost and heuristic estimation, leading to an efficient and optimal solution.





- Key Concepts :-

  A* search uses the evaluation function:

  f(n)=g(n)+h(n)

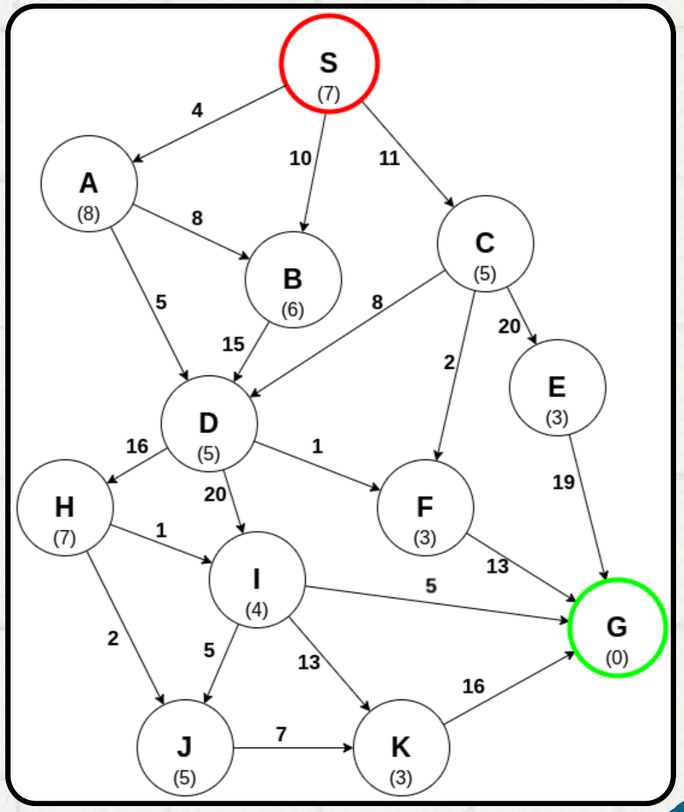
  f(n): Total cost from start to goal through node n.

  g(n): Cost from the start node to node n.

  h(n): Heuristic estimate of cost from node n to the goal.

A* working

1. **Initialization:** Start with the initial node in the priority queue with f(n)=g(n)+h(n).
2. **Node Expansion:** Extract the node with the lowest f(n) value. If it's the goal node, the search ends.
3. **Neighbor Evaluation:** For each neighbor of the current node, compute the tentative g(n). If it offers a better path, update g(n) and f(n), then add it to the queue.
4. **Repeat:** Continue expanding nodes until the goal is reached or the queue is empty.



Thank you!