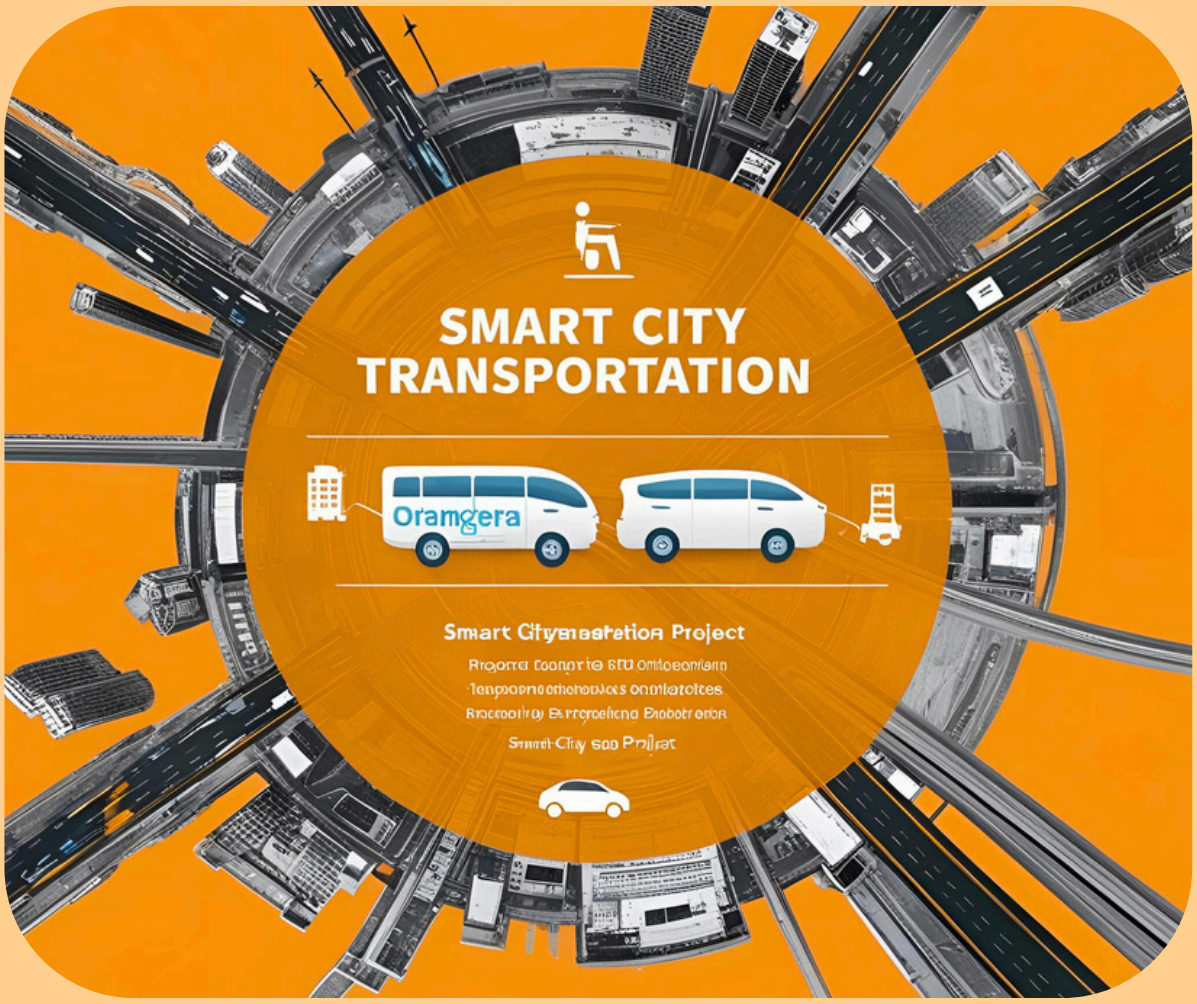
AIU.
ALAMEIN
INTERNATIONAL UNIVERSITY
جامعـة العلميـن الدوليـة



- **Abdulrhman Ibrahim** 22101286
- **Hazem Oraby** 22101052
- **Wafaa Muhammed** 22101116
- **Rokiaa Islam** 22101084
- **Tasneem Ibrahim** 22101382

System Architecture &Design

The architecture of the Smart City Transportation Optimization System is divided into four main layers:

1-**User Interface (UI):**
This layer allows the user to interact with the system by selecting start and end points and viewing the optimal path on a map.

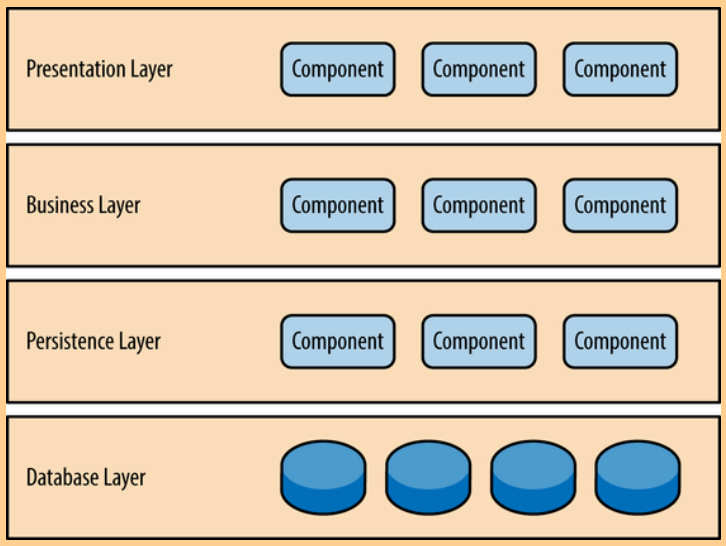
2.Application Logic / Algorithm Layer:
This is the core layer where all the main algorithms are implemented. It includes:

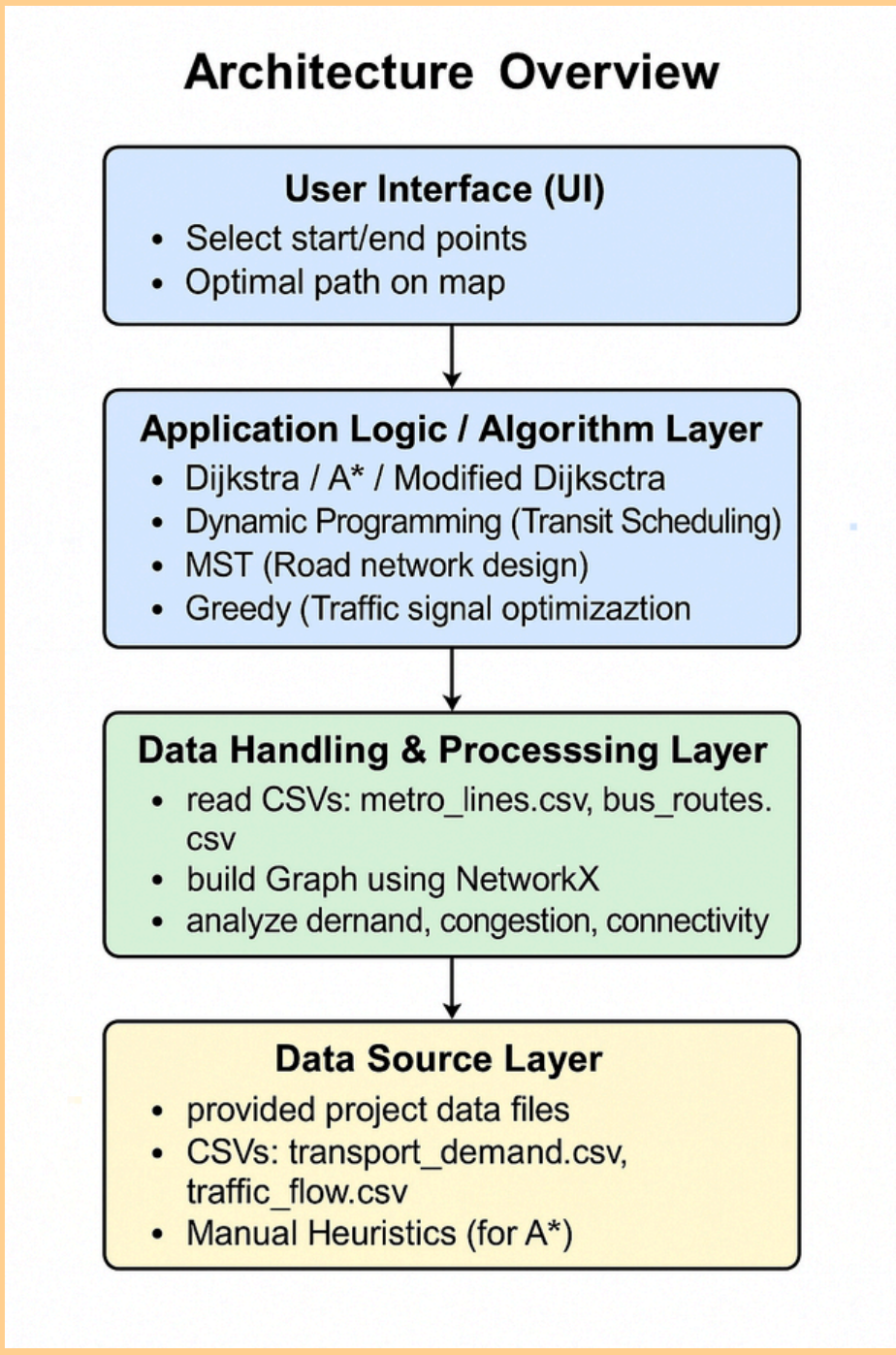
- **Dijkstra's algorithm for shortest path**
- **A* algorithm for emergency routing**
- **Modified Dijkstra to handle traffic conditions**
- **Dynamic Programming for transit scheduling**
- **MST (Minimum Spanning Tree) for road network design**
- **Greedy algorithms for traffic signal optimization**

3-Data Handling & Processing Layer:
This layer is responsible for reading and processing the input data (CSV files). It builds the graph using NetworkX and analyzes demand, congestion, and connectivity.

4-Data Source Layer:
The lowest layer contains the raw data used by the system, such as:

- **metro_lines.csv**
- **bus_routes.csv**
- **transport_demand.csv**
- **traffic_flow.csv**
- **It also includes manual heuristics used for A* search.**





Algorithm Implementations & Analyses

* Minimum Spanning Tree (MST)

- **Algorithm**: Kruskal's algorithm was used to connect high-priority areas while minimizing road construction cost.
- **Modification**: Critical facilities (e.g., hospitals, airports) were prioritized using weighted edges and filtered node sets.

* Shortest Path Algorithm

- **Algorithm:** Dijkstra's algorithm with priority queue (min-heap).
- **Use Case:** Optimal routing between any two districts of Cairo.
- **Traffic-Aware:** Weights were dynamically adjusted to simulate peak vs. off-peak hours.

* Dynamic Programming (Transit Scheduling)

- **Approach:** Custom time-expanded simulation to estimate wait + travel time based on vehicle frequency.
- **Assumption:** Each bus and metro line has fixed frequency; wait time = frequency - (current time % frequency).

* Greedy Algorithm

- **Idea: Minimize waiting time at intersections based on incoming vehicle density (placeholder logic not fully implemented in prototype).**

Complexity Analysis

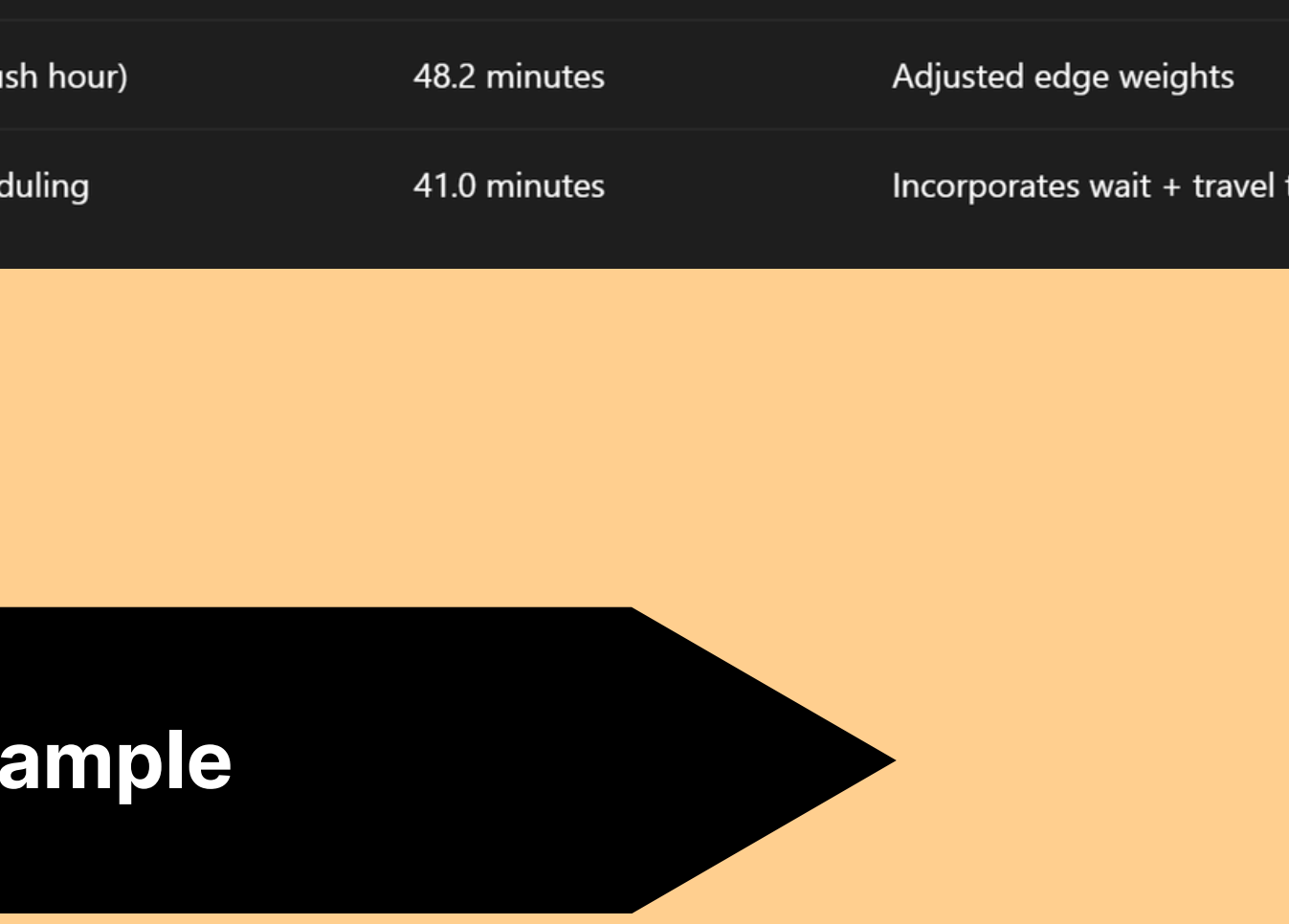
Algorithm	Time Complexity	Notes
MST (Kruskal)	O(E log V)	Union-Find structure used
Dijkstra	O(E + V log V)	With min-heap (priority queue)
A*	O(E) – O(E log V)	Depends on heuristic quality
DP Scheduling	Depends on problem size	Similar to 0/1 Knapsack
Greedy	O(n log n) or O(n)	Depending on implementation

Performance Evaluation

✅ **Results:**

Scenario	Avg Travel Time	Notes
Static Dijkstra	35.6 minutes	Does not account for traffic
Dynamic Dijkstra (rush hour)	48.2 minutes	Adjusted edge weights
Transit + Time Scheduling	41.0 minutes	Incorporates wait + travel times

Demo Example



Challenges and Solutions

Challenge	Solution
Handling missing or unconnected nodes	Added validation before pathfinding
Real-time traffic data integration	Used dynamic weights based on time
Optimizing schedule distribution	Applied DP and adjusted weights

Future Improvements

- **Build a real-time GUI (Web or Mobile app).**
- **Integrate GPS/live data feeds for traffic prediction.**
- **Use Machine Learning for demand forecasting and dynamic optimization.**