**Project Proposal: GIKAB – A Ride-Hailing App for GIKI**

**Course:** Data Structures & Algorithms (CS-221)  
**Team Members:** Syed Muhammad Ali Akbar Shah (2024615), Omer Imran Siddiqui (2024523), Muhammad Danish (2024372)

## 1. Project Overview and Problem Statement

**Problem:**

In a university environment like GIKI, students and staff often face difficulties in coordinating rides around campus and nearby locations. There is currently no unified system for ride-sharing or transportation within GIKI, leading to inefficient travel, time wastage, and communication issues.

**Proposed Solution:**

We aim to develop an online ride-hailing app called **GIKAB**, tailored for GIKI students and faculty. GIKAB will function similarly to Uber but on a smaller, campus-based scale. The platform will allow users to request rides, view available drivers, manage rides, and track ride history.

The primary goal is to build this system in C++ using data structures and algorithms learned in class, emphasizing real-world applications. A basic GUI may be added using **Qt** if time allows.

## 2. Key Data Structures to be Implemented

We will implement and integrate the following core data structures:

| **Data Structure** | **Use Case** |
| --- | --- |
| Linked List | - Maintain dynamic list of available drivers and riders - Store user ride history |
| Queue | - Handle ride requests (First-Come, First-Served basis) - Match riders to drivers in order |
| Stack | - Allow "undo" operation for recent actions (e.g., ride cancellations) - Navigation history |
| Binary Tree (if time permits) | - Organize and search driver data efficiently (e.g., by ratings or availability time) |

We chose these structures to ensure efficiency, scalability, and hands-on practice with dynamic data manipulation.

## 3. Core Algorithms and Their Application

| **Algorithm** | **Purpose** |
| --- | --- |
| **Matching Algorithm** | Match riders to nearest or first available drivers from the queue. |
| **Search Algorithm** | Search for drivers/riders by ID, name, or location (implemented using binary search if using a tree). |
| **Sorting Algorithm** | Sort drivers by rating or time of availability (for displaying options). |
| **Traversal Algorithms** | If using binary trees, we will implement in-order, pre-order, and post-order traversals to display/search nodes. |

These algorithms will be optimized for speed and clarity, ensuring a smooth experience.

## 4. Data Flow: Input, Processing, and Output

**Input (Data Reception):**

* Users will input:
  + Name, ID, role (driver/rider)
  + Ride request or availability
  + Ratings (after ride)
* Inputs received via CLI initially; GUI input forms if Qt is implemented.

**Processing:**

* Store ride requests in **queue**.
* Manage available drivers using a **linked list**.
* Match riders to drivers using a **matching algorithm**.
* Allow ride cancellation using **stack** for undo feature.
* Store completed rides into **history linked list**.
* Ratings may be stored and organized in a **binary tree** if added.

**Output (Display):**

* Display ride statuses: waiting, matched, completed
* Show list of drivers or ride history
* GUI output using Qt (optional), otherwise via terminal

## 5. Integration of Data Structures & Algorithms Concepts

This project directly integrates key concepts from our course:

| **Concept** | **Integration** |
| --- | --- |
| **Linked List** | Dynamic user data and ride history |
| **Queue** | Ride request management |
| **Stack** | Undo actions, navigation history |
| **Trees** | Efficient driver search and organization |
| **Algorithm Design** | Matching, searching, sorting |
| **Modular Design** | Each structure and logic will be implemented in separate modules/classes |
| **Object-Oriented Programming** | Drivers, Riders, and Rides will be classes using encapsulation and inheritance where applicable |

The focus will be on demonstrating practical implementation of abstract data structures and creating a usable product to reflect our understanding.