Q1)

C/C++

```
/*
 * N-Queens Problem Solution
 * Author: ADIANDRA NALLAKKA GARI SUPRITHA
 * This code demonstrates the classic N-Queens backtracking algorithm
 * Written with optimized conflict detection for educational purposes
 */

#include <iostream>
#include <vector>
#include <string>
using namespace std;

class NQueensSolver {
private:
    vector<vector<string>> solutions;
    vector<bool> column_used;       // Track occupied columns
    vector<bool> main_diagonal;     // Track main diagonals (i-j+n-1 as index)
    vector<bool> anti_diagonal;     // Track anti-diagonals (i+j as index)

    void solve_recursive(vector<string>& chessboard, int current_row, int
board_size) {
        // Base case: successfully placed all queens
        if (current_row == board_size) {
            solutions.push_back(chessboard);
            return;
        }

        // Try placing queen in each column of current row
        for (int current_col = 0; current_col < board_size; current_col++) {
            // Check if current position conflicts with existing queens
            if (column_used[current_col] ||
                main_diagonal[current_row - current_col + board_size - 1] ||
                anti_diagonal[current_row + current_col]) {
                continue; // Skip this position due to conflict
            }

            // Place the queen at current position
            chessboard[current_row][current_col] = 'Q';
            column_used[current_col] = true;
            main_diagonal[current_row - current_col + board_size - 1] = true;
            anti_diagonal[current_row + current_col] = true;
```

```
            // Move to next row
            solve_recursive(chessboard, current_row + 1, board_size);

            // Backtrack: remove the queen and reset flags
            chessboard[current_row][current_col] = '.';
            column_used[current_col] = false;
            main_diagonal[current_row - current_col + board_size - 1] = false;
            anti_diagonal[current_row + current_col] = false;
        }
    }

public:
    vector<vector<string>> findAllSolutions(int n) {
        // Reset solutions for fresh start
        solutions.clear();

        // Initialize conflict tracking arrays
        column_used.assign(n, false);
        main_diagonal.assign(2 * n - 1, false);    // For main diagonals
        anti_diagonal.assign(2 * n - 1, false);    // For anti-diagonals

        // Create empty chessboard
        vector<string> chessboard(n, string(n, '.'));

        // Begin solving from first row
        solve_recursive(chessboard, 0, n);

        return solutions;
    }
};

// Function to display all solutions in a readable format
void display_all_solutions(const vector<vector<string>>& all_solutions) {
    cout << "Number of solutions found: " << all_solutions.size() << "\n\n";

    for (int solution_num = 0; solution_num < all_solutions.size();
solution_num++) {
        cout << "Configuration " << (solution_num + 1) << ":\n";
        for (const string& board_row : all_solutions[solution_num]) {
            cout << board_row << "\n";
        }
        cout << "\n";
    }
```

```
}

int main() {
    // Code written by ADIANDRA NALLAKKA GARI SUPRITHA
    cout << "/*****************************************\n";
    cout << " * N-Queens Puzzle Solver                 *\n";
    cout << " * Author: ADIANDRA NALLAKKA GARI SUPRITHA  *\n";
    cout << " * Backtracking Algorithm Implementation    *\n";
    cout << " *****************************************/\n\n";

    NQueensSolver puzzle_solver;

    // Test case 1: 4x4 board
    cout << "=== Solving N-Queens for 4x4 board ===\n";
    vector<vector<string>> solutions_4x4 = puzzle_solver.findAllSolutions(4);
    display_all_solutions(solutions_4x4);

    // Test case 2: 1x1 board (trivial case)
    cout << "=== Solving N-Queens for 1x1 board ===\n";
    vector<vector<string>> solutions_1x1 = puzzle_solver.findAllSolutions(1);
    display_all_solutions(solutions_1x1);

    // Test case 3: 8x8 board (classic problem)
    cout << "=== Solving N-Queens for 8x8 board ===\n";
    vector<vector<string>> solutions_8x8 = puzzle_solver.findAllSolutions(8);
    cout << "Total configurations for 8x8 board: " << solutions_8x8.size() <<
"\n";

    // Display first configuration for 8x8
    if (!solutions_8x8.empty()) {
        cout << "\nFirst valid configuration for 8x8:\n";
        for (const string& board_row : solutions_8x8[0]) {
            cout << board_row << "\n";
        }
    }

    // Interactive section for user input
    cout << "\n=== Custom Board Size ===\n";
    int user_input;
    cout << "Please enter board size (between 1 and 9): ";
    cin >> user_input;

    if (user_input >= 1 && user_input <= 9) {
```

```
        vector<vector<string>> custom_solutions =
puzzle_solver.findAllSolutions(user_input);
        cout << "\n=== Solutions for " << user_input << "x" << user_input << "
board ===\n";
        display_all_solutions(custom_solutions);

        cout << "Algorithm execution completed successfully!\n";
        cout << "Code developed by: ADIANDRA NALLAKKA GARI SUPRITHA\n";
    } else {
        cout << "Error: Board size must be between 1 and 9!\n";
    }

    return 0;
}
```

Q2)

```
C/C++
/*
* Circular Dependency Detection in Module Loading System
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
* This code implements cycle detection in directed graphs using DFS
* Written for dependency management in large software systems
*/

#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

class DependencyChecker {
private:
   vector<vector<int>> adjacency_graph;
   vector<int> visit_status;  // 0 = unvisited, 1 = visiting, 2 = visited

   bool detect_cycle_dfs(int current_module) {
       // Mark current module as being visited (in recursion stack)
       visit_status[current_module] = 1;
```

```
        // Check all dependencies of current module
        for (int dependent_module : adjacency_graph[current_module]) {
            if (visit_status[dependent_module] == 1) {
                // Found a back edge - cycle detected!
                return true;
            }

            // If module not visited yet, recursively check for cycles
            if (visit_status[dependent_module] == 0 &&
                detect_cycle_dfs(dependent_module)) {
                return true;
            }
        }

        // Mark current module as completely processed
        visit_status[current_module] = 2;
        return false;
    }

public:
    bool has_circular_dependency(int total_modules, vector<vector<int>>&
dependency_edges) {
        // Initialize adjacency list representation
        adjacency_graph.assign(total_modules, vector<int>());
        visit_status.assign(total_modules, 0);

        // Build the dependency graph
        for (const auto& edge_pair : dependency_edges) {
            int module_from = edge_pair[0];
            int module_to = edge_pair[1];
            adjacency_graph[module_from].push_back(module_to);
        }

        // Check each module for cycles (handles disconnected components)
        for (int module_id = 0; module_id < total_modules; module_id++) {
            if (visit_status[module_id] == 0) {
                if (detect_cycle_dfs(module_id)) {
                    return true;  // Circular dependency found
                }
            }
        }

        return false;  // No circular dependencies detected
    }
```

```
};

// Function to display the dependency graph structure
void display_dependency_graph(int total_modules, const vector<vector<int>>&
dependency_edges) {
   cout << "Dependency Graph Structure:\n";
   cout << "Total Modules: " << total_modules << "\n";
   cout << "Dependencies (module -> depends on):\n";

   for (const auto& edge_pair : dependency_edges) {
       cout << "Module " << edge_pair[0] << " depends on Module " <<
edge_pair[1] << "\n";
   }
   cout << "\n";
}

// Function to run test cases and display results
void run_test_case(int test_number, int total_modules, vector<vector<int>>
dependency_edges) {
   cout << "=== Test Case " << test_number << " ===\n";
   display_dependency_graph(total_modules, dependency_edges);

   DependencyChecker module_checker;
   bool has_cycle = module_checker.has_circular_dependency(total_modules,
dependency_edges);

   cout << "Result: " << (has_cycle ? "CIRCULAR DEPENDENCY DETECTED" : "NO
CIRCULAR DEPENDENCY") << "\n";
   cout << "Status: " << (has_cycle ? "UNSAFE TO LOAD" : "SAFE TO LOAD") <<
"\n\n";
}

int main() {
   // Code written by ADIANDRA NALLAKKA GARI SUPRITHA
   cout << "/*********************************************************\n";
   cout << " * Module Dependency Cycle Detection System             *\n";
   cout << " * Author: ADIANDRA NALLAKKA GARI SUPRITHA               *\n";
   cout << " * DFS-based Circular Dependency Detection Algorithm    *\n";
   cout << " *********************************************************/\n\n";

   // Test Case 1: Linear dependency chain (no cycle)
   vector<vector<int>> test_case_1 = {{0, 1}, {1, 2}, {2, 3}};
   run_test_case(1, 4, test_case_1);
```

```
// Test Case 2: Circular dependency (has cycle)
vector<vector<int>> test_case_2 = {{0, 1}, {1, 2}, {2, 0}};
run_test_case(2, 4, test_case_2);

// Test Case 3: Self-dependency (cycle with single node)
vector<vector<int>> test_case_3 = {{0, 0}, {1, 2}};
run_test_case(3, 3, test_case_3);

// Test Case 4: Complex dependency with multiple components
vector<vector<int>> test_case_4 = {{0, 1}, {1, 2}, {3, 4}, {4, 5}, {5, 3}};
run_test_case(4, 6, test_case_4);

// Test Case 5: No dependencies (isolated modules)
vector<vector<int>> test_case_5 = {};
run_test_case(5, 3, test_case_5);

// Interactive section for custom input
cout << "=== Custom Dependency Check ===\n";
int user_modules;
cout << "Enter number of modules: ";
cin >> user_modules;

if (user_modules < 1 || user_modules > 10000) {
    cout << "Error: Number of modules must be between 1 and 10000!\n";
    return 1;
}

int dependency_count;
cout << "Enter number of dependencies: ";
cin >> dependency_count;

if (dependency_count < 0 || dependency_count > 100000) {
    cout << "Error: Number of dependencies must be between 0 and 100000!\n";
    return 1;
}

vector<vector<int>> user_dependencies;
cout << "Enter dependencies (format: module_from module_to):\n";

for (int i = 0; i < dependency_count; i++) {
    int module_from, module_to;
    cout << "Dependency " << (i + 1) << ": ";
    cin >> module_from >> module_to;
```

```
        if (module_from < 0 || module_from >= user_modules ||
            module_to < 0 || module_to >= user_modules) {
            cout << "Error: Module IDs must be between 0 and " << (user_modules
- 1) << "!\n";
            return 1;
        }

        user_dependencies.push_back({module_from, module_to});
    }

    cout << "\n=== Analysis of Your Dependency Graph ===\n";
    display_dependency_graph(user_modules, user_dependencies);

    DependencyChecker custom_checker;
    bool custom_result = custom_checker.has_circular_dependency(user_modules,
user_dependencies);

    cout << "Final Analysis: " << (custom_result ? "CIRCULAR DEPENDENCY
DETECTED!" : "NO CIRCULAR DEPENDENCY FOUND") << "\n";
    cout << "Recommendation: " << (custom_result ? "Fix circular dependencies
before loading modules" : "Safe to proceed with module loading") << "\n";

    cout << "\nAlgorithm execution completed successfully!\n";
    cout << "Developed by: ADIANDRA NALLAKKA GARI SUPRITHA\n";

    return 0;
}

// Global function matching the required signature
bool hasCircularDependency(int n, vector<vector<int>>& edges) {
    /*
     * Author: ADIANDRA NALLAKKA GARI SUPRITHA
     * Required function signature implementation
     */
    DependencyChecker checker;
    return checker.has_circular_dependency(n, edges);
}
```

Q3)

```C/C++
/*
* GPU-Accelerated Particle System - Fireworks Display
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
* OpenGL ES 3.0 implementation with advanced shader programming
* Optimized for high-performance particle rendering
*/

#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <iostream>
#include <vector>
#include <random>
#include <cmath>
#include <string>

using namespace glm;
using namespace std;

// Particle structure for CPU-side management
struct ParticleData {
   vec3 position;
   vec3 velocity;
   vec4 color;
   float lifetime;
   float max_lifetime;
   bool is_active;
};

class FireworksParticleSystem {
private:
   // OpenGL resources
   GLuint shader_program;
   GLuint vertex_array_object;
   GLuint vertex_buffer_object;
   GLuint instance_buffer_object;

   // Shader uniform locations
   GLint uniform_projection_matrix;
   GLint uniform_view_matrix;
```

```
GLint uniform_particle_size;
GLint uniform_time_elapsed;

// Particle system parameters
static const int MAX_PARTICLE_COUNT = 5000;
vector<ParticleData> particle_pool;
vector<vec3> instance_positions;
vector<vec4> instance_colors;
vector<float> instance_sizes;

int active_particle_count;
float gravity_strength;
float particle_base_size;

// Random number generation
mt19937 random_generator;
uniform_real_distribution<float> color_distribution;
uniform_real_distribution<float> velocity_distribution;
uniform_real_distribution<float> angle_distribution;

// Timing and interaction
double previous_frame_time;
vec2 last_click_position;
bool should_create_burst;

// Vertex shader source code
const char* vertex_shader_source = R"(
    #version 330 core

    // Per-vertex attributes
    layout (location = 0) in vec3 vertex_position;

    // Per-instance attributes
    layout (location = 1) in vec3 particle_position;
    layout (location = 2) in vec4 particle_color;
    layout (location = 3) in float particle_size;

    // Uniforms
    uniform mat4 projection_matrix;
    uniform mat4 view_matrix;
    uniform float time_elapsed;

    // Output to fragment shader
    out vec4 fragment_color;
```

```
        out vec2 texture_coords;

        void main() {
            // Calculate final position with particle offset
            vec3 world_position = vertex_position * particle_size +
particle_position;

            // Apply view and projection transformations
            gl_Position = projection_matrix * view_matrix * vec4(world_position,
1.0);

            // Pass color and texture coordinates to fragment shader
            fragment_color = particle_color;
            texture_coords = vertex_position.xy + 0.5; // Convert from [-0.5,
0.5] to [0, 1]

            // Add subtle pulsing effect based on time
            float pulse_factor = 1.0 + 0.2 * sin(time_elapsed * 3.0);
            gl_PointSize = particle_size * pulse_factor;
        }
    )";

    // Fragment shader source code
    const char* fragment_shader_source = R"(
        #version 330 core

        // Input from vertex shader
        in vec4 fragment_color;
        in vec2 texture_coords;

        // Output color
        out vec4 final_color;

        void main() {
            // Create circular particle shape
            vec2 center_offset = texture_coords - vec2(0.5, 0.5);
            float distance_from_center = length(center_offset);

            // Smooth circular falloff
            float alpha_falloff = 1.0 - smoothstep(0.0, 0.5,
distance_from_center);

            // Add sparkle effect
```

```
            float sparkle_intensity = 1.0 + 0.5 * sin(distance_from_center *
20.0);

            // Combine color with alpha and sparkle
            final_color = vec4(fragment_color.rgb * sparkle_intensity,
                               fragment_color.a * alpha_falloff);

            // Discard completely transparent fragments
            if (final_color.a < 0.01) {
                discard;
            }
        }
    )";

public:
    FireworksParticleSystem() :
        active_particle_count(0),
        gravity_strength(9.8f),
        particle_base_size(8.0f),

random_generator(chrono::steady_clock::now().time_since_epoch().count()),
        color_distribution(0.0f, 1.0f),
        velocity_distribution(-15.0f, 15.0f),
        angle_distribution(0.0f, 2.0f * M_PI),
        previous_frame_time(0.0),
        should_create_burst(false) {

        // Initialize particle pool
        particle_pool.resize(MAX_PARTICLE_COUNT);
        instance_positions.resize(MAX_PARTICLE_COUNT);
        instance_colors.resize(MAX_PARTICLE_COUNT);
        instance_sizes.resize(MAX_PARTICLE_COUNT);

        for (int i = 0; i < MAX_PARTICLE_COUNT; i++) {
            particle_pool[i].is_active = false;
        }
    }

    bool initialize_opengl_resources() {
        // Compile and link shaders
        if (!create_shader_program()) {
            cout << "Failed to create shader program!\n";
            return false;
        }
```

```
        // Create vertex array and buffers
        create_vertex_buffers();

        // Get uniform locations
        uniform_projection_matrix = glGetUniformLocation(shader_program,
"projection_matrix");
        uniform_view_matrix = glGetUniformLocation(shader_program,
"view_matrix");
        uniform_particle_size = glGetUniformLocation(shader_program,
"particle_size");
        uniform_time_elapsed = glGetUniformLocation(shader_program,
"time_elapsed");

        // Enable blending for additive effects
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE); // Additive blending

        // Enable depth testing but disable depth writing for particles
        glEnable(GL_DEPTH_TEST);
        glDepthMask(GL_FALSE);

        cout << "OpenGL particle system initialized successfully!\n";
        cout << "Author: ADIANDRA NALLAKKA GARI SUPRITHA\n";
        return true;
    }

    void create_fireworks_burst(vec2 screen_position, int particle_count = 800)
{
        // Convert screen coordinates to world coordinates
        vec3 burst_center = vec3(screen_position.x, screen_position.y, 0.0f);

        int particles_created = 0;
        for (int i = 0; i < MAX_PARTICLE_COUNT && particles_created <
particle_count; i++) {
            if (!particle_pool[i].is_active) {
                // Initialize particle properties
                ParticleData& new_particle = particle_pool[i];

                // Random explosion direction
                float explosion_angle = angle_distribution(random_generator);
                float explosion_speed = velocity_distribution(random_generator)
+ 20.0f;
```

```
                new_particle.position = burst_center;
                new_particle.velocity = vec3(
                    cos(explosion_angle) * explosion_speed,
                    sin(explosion_angle) * explosion_speed + 10.0f, // Add
upward bias
                    velocity_distribution(random_generator) * 0.5f
                );

                // Random vibrant colors
                float hue_shift = color_distribution(random_generator);
                new_particle.color = vec4(
                    0.8f + 0.2f * sin(hue_shift * 6.0f),
                    0.6f + 0.4f * cos(hue_shift * 4.0f),
                    0.9f + 0.1f * sin(hue_shift * 8.0f),
                    1.0f
                );

                new_particle.lifetime = 3.0f +
color_distribution(random_generator) * 2.0f;
                new_particle.max_lifetime = new_particle.lifetime;
                new_particle.is_active = true;

                particles_created++;
            }
        }

        cout << "Created fireworks burst with " << particles_created << "
particles at ("
             << screen_position.x << ", " << screen_position.y << ")\n";
    }

    void update_particle_system(float delta_time) {
        active_particle_count = 0;

        for (int i = 0; i < MAX_PARTICLE_COUNT; i++) {
            ParticleData& current_particle = particle_pool[i];

            if (current_particle.is_active) {
                // Update particle physics
                current_particle.velocity.y -= gravity_strength * delta_time;
                current_particle.position += current_particle.velocity *
delta_time;

                // Update lifetime
```

```
                current_particle.lifetime -= delta_time;

                if (current_particle.lifetime <= 0.0f) {
                    current_particle.is_active = false;
                    continue;
                }

                // Calculate fade factor
                float life_ratio = current_particle.lifetime /
current_particle.max_lifetime;
                float fade_alpha = smoothstep(0.0f, 0.3f, life_ratio);

                // Update instance data for rendering
                instance_positions[active_particle_count] =
current_particle.position;
                instance_colors[active_particle_count] = vec4(
                    current_particle.color.rgb,
                    current_particle.color.a * fade_alpha
                );
                instance_sizes[active_particle_count] = particle_base_size *
(1.0f + life_ratio);

                active_particle_count++;
            }
        }

        // Update GPU buffers with new instance data
        update_instance_buffers();
    }

    void render_particles(mat4 projection_matrix, mat4 view_matrix, float
current_time) {
        if (active_particle_count == 0) return;

        glUseProgram(shader_program);

        // Set uniform matrices
        glUniformMatrix4fv(uniform_projection_matrix, 1, GL_FALSE,
value_ptr(projection_matrix));
        glUniformMatrix4fv(uniform_view_matrix, 1, GL_FALSE,
value_ptr(view_matrix));
        glUniform1f(uniform_time_elapsed, current_time);

        // Bind vertex array and render instances
```

```
        glBindVertexArray(vertex_array_object);
        glDrawArraysInstanced(GL_TRIANGLES, 0, 6, active_particle_count);
        glBindVertexArray(0);

        glUseProgram(0);
    }

    void handle_mouse_click(double mouse_x, double mouse_y, int window_width,
int window_height) {
        // Convert mouse coordinates to normalized device coordinates
        float normalized_x = (2.0f * mouse_x / window_width) - 1.0f;
        float normalized_y = 1.0f - (2.0f * mouse_y / window_height);

        // Scale to world coordinates
        vec2 world_position = vec2(normalized_x * 400.0f, normalized_y *
300.0f);

        create_fireworks_burst(world_position);
    }

    int get_active_particle_count() const {
        return active_particle_count;
    }

    void cleanup_resources() {
        glDeleteVertexArrays(1, &vertex_array_object);
        glDeleteBuffers(1, &vertex_buffer_object);
        glDeleteBuffers(1, &instance_buffer_object);
        glDeleteProgram(shader_program);

        cout << "Particle system resources cleaned up successfully!\n";
        cout << "Code developed by: ADIANDRA NALLAKKA GARI SUPRITHA\n";
    }

private:
    bool create_shader_program() {
        // Compile vertex shader
        GLuint vertex_shader = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vertex_shader, 1, &vertex_shader_source, nullptr);
        glCompileShader(vertex_shader);

        if (!check_shader_compilation(vertex_shader, "VERTEX")) {
            return false;
        }
```

```
    // Compile fragment shader
    GLuint fragment_shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragment_shader, 1, &fragment_shader_source, nullptr);
    glCompileShader(fragment_shader);

    if (!check_shader_compilation(fragment_shader, "FRAGMENT")) {
        return false;
    }

    // Link shader program
    shader_program = glCreateProgram();
    glAttachShader(shader_program, vertex_shader);
    glAttachShader(shader_program, fragment_shader);
    glLinkProgram(shader_program);

    if (!check_program_linking(shader_program)) {
        return false;
    }

    // Clean up individual shaders
    glDeleteShader(vertex_shader);
    glDeleteShader(fragment_shader);

    return true;
}

void create_vertex_buffers() {
    // Quad vertices for particle sprites
    float quad_vertices[] = {
        -0.5f, -0.5f, 0.0f,
         0.5f, -0.5f, 0.0f,
         0.5f,  0.5f, 0.0f,
        -0.5f, -0.5f, 0.0f,
         0.5f,  0.5f, 0.0f,
        -0.5f,  0.5f, 0.0f
    };

    // Create and bind vertex array object
    glGenVertexArrays(1, &vertex_array_object);
    glBindVertexArray(vertex_array_object);

    // Create vertex buffer for quad geometry
    glGenBuffers(1, &vertex_buffer_object);
```

```
        glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_object);
        glBufferData(GL_ARRAY_BUFFER, sizeof(quad_vertices), quad_vertices,
GL_STATIC_DRAW);

        // Set vertex attribute pointers
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
        glEnableVertexAttribArray(0);

        // Create instance buffer for particle data
        glGenBuffers(1, &instance_buffer_object);
        glBindBuffer(GL_ARRAY_BUFFER, instance_buffer_object);
        glBufferData(GL_ARRAY_BUFFER,
                     MAX_PARTICLE_COUNT * (sizeof(vec3) + sizeof(vec4) +
sizeof(float)),
                     nullptr, GL_DYNAMIC_DRAW);

        // Setup instance attributes
        setup_instance_attributes();

        glBindVertexArray(0);
    }

    void setup_instance_attributes() {
        GLsizei stride = sizeof(vec3) + sizeof(vec4) + sizeof(float);

        // Position attribute (location 1)
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, stride, (void*)0);
        glEnableVertexAttribArray(1);
        glVertexAttribDivisor(1, 1);

        // Color attribute (location 2)
        glVertexAttribPointer(2, 4, GL_FLOAT, GL_FALSE, stride,
(void*)sizeof(vec3));
        glEnableVertexAttribArray(2);
        glVertexAttribDivisor(2, 1);

        // Size attribute (location 3)
        glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, stride,
                              (void*)(sizeof(vec3) + sizeof(vec4)));
        glEnableVertexAttribArray(3);
        glVertexAttribDivisor(3, 1);
    }
```

```
    void update_instance_buffers() {
        glBindBuffer(GL_ARRAY_BUFFER, instance_buffer_object);

        // Update positions
        glBufferSubData(GL_ARRAY_BUFFER, 0,
                        active_particle_count * sizeof(vec3),
                        instance_positions.data());

        // Update colors
        glBufferSubData(GL_ARRAY_BUFFER,
                        MAX_PARTICLE_COUNT * sizeof(vec3),
                        active_particle_count * sizeof(vec4),
                        instance_colors.data());

        // Update sizes
        glBufferSubData(GL_ARRAY_BUFFER,
                        MAX_PARTICLE_COUNT * (sizeof(vec3) + sizeof(vec4)),
                        active_particle_count * sizeof(float),
                        instance_sizes.data());

        glBindBuffer(GL_ARRAY_BUFFER, 0);
    }

    bool check_shader_compilation(GLuint shader_id, const string& shader_type) {
        GLint compilation_success;
        GLchar info_log[1024];

        glGetShaderiv(shader_id, GL_COMPILE_STATUS, &compilation_success);
        if (!compilation_success) {
            glGetShaderInfoLog(shader_id, 1024, nullptr, info_log);
            cout << "ERROR: " << shader_type << " shader compilation failed!\n"
<< info_log << "\n";
            return false;
        }
        return true;
    }

    bool check_program_linking(GLuint program_id) {
        GLint linking_success;
        GLchar info_log[1024];

        glGetProgramiv(program_id, GL_LINK_STATUS, &linking_success);
        if (!linking_success) {
            glGetProgramInfoLog(program_id, 1024, nullptr, info_log);
```

```
            cout << "ERROR: Shader program linking failed!\n" << info_log <<
"\n";
            return false;
        }
        return true;
    }
};

// Global variables for GLFW callbacks
FireworksParticleSystem* global_particle_system = nullptr;

void mouse_button_callback(GLFWwindow* window, int button, int action, int
mods) {
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS &&
global_particle_system) {
        double mouse_x, mouse_y;
        glfwGetCursorPos(window, &mouse_x, &mouse_y);

        int window_width, window_height;
        glfwGetWindowSize(window, &window_width, &window_height);

        global_particle_system->handle_mouse_click(mouse_x, mouse_y,
window_width, window_height);
    }
}

void key_callback(GLFWwindow* window, int key, int scancode, int action, int
mods) {
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS) {
        glfwSetWindowShouldClose(window, GLFW_TRUE);
    }

    if (key == GLFW_KEY_SPACE && action == GLFW_PRESS && global_particle_system)
{
        // Create random burst
        global_particle_system->create_fireworks_burst(
            vec2((rand() % 800) - 400, (rand() % 600) - 300), 1000
        );
    }
}

int main() {
    cout << "/************************************************************\n";
    cout << " * GPU-Accelerated Fireworks Particle System                *\n";
```

```
    cout << " * Author: ADIANDRA NALLAKKA GARI SUPRITHA                *\n";
    cout << " * OpenGL ES 3.0 + Advanced Shader Programming            *\n";
    cout << " * Click to create fireworks, Space for random bursts     *\n";
    cout << "
************************************************************/\n\n";

    // Initialize GLFW
    if (!glfwInit()) {
        cout << "Failed to initialize GLFW!\n";
        return -1;
    }

    // Configure GLFW
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_SAMPLES, 4); // Enable multisampling

    // Create window
    GLFWwindow* main_window = glfwCreateWindow(1200, 800,
        "Fireworks Particle System - by ADIANDRA NALLAKKA GARI SUPRITHA",
        nullptr, nullptr);

    if (!main_window) {
        cout << "Failed to create GLFW window!\n";
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(main_window);

    // Initialize GLEW
    if (glewInit() != GLEW_OK) {
        cout << "Failed to initialize GLEW!\n";
        return -1;
    }

    // Setup callbacks
    glfwSetMouseButtonCallback(main_window, mouse_button_callback);
    glfwSetKeyCallback(main_window, key_callback);

    // Initialize particle system
    FireworksParticleSystem particle_system;
    global_particle_system = &particle_system;
```

```
    if (!particle_system.initialize_opengl_resources()) {
        return -1;
    }

    // Setup projection matrix
    mat4 projection_matrix = ortho(-600.0f, 600.0f, -400.0f, 400.0f, -100.0f,
100.0f);
    mat4 view_matrix = lookAt(vec3(0, 0, 50), vec3(0, 0, 0), vec3(0, 1, 0));

    // Create initial welcome burst
    particle_system.create_fireworks_burst(vec2(0.0f, 0.0f), 1200);

    double last_frame_time = glfwGetTime();

    // Main rendering loop
    while (!glfwWindowShouldClose(main_window)) {
        double current_frame_time = glfwGetTime();
        float delta_time = static_cast<float>(current_frame_time -
last_frame_time);
        last_frame_time = current_frame_time;

        // Handle events
        glfwPollEvents();

        // Update particle system
        particle_system.update_particle_system(delta_time);

        // Clear screen with dark background
        glClearColor(0.05f, 0.05f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Render particles
        particle_system.render_particles(projection_matrix, view_matrix,
                                         static_cast<float>(current_frame_time));

        // Display performance info in title
        if (static_cast<int>(current_frame_time) % 2 == 0) {
            string performance_title = "Fireworks System - Active Particles: " +

to_string(particle_system.get_active_particle_count()) +
                                       " - by ADIANDRA NALLAKKA GARI SUPRITHA";
            glfwSetWindowTitle(main_window, performance_title.c_str());
        }
```

```
        glfwSwapBuffers(main_window);
    }

    // Cleanup
    particle_system.cleanup_resources();
    glfwTerminate();

    cout << "\nFireworks particle system demonstration completed!\n";
    cout << "Thank you for experiencing this GPU-accelerated simulation!\n";
    cout << "Developed with passion by: ADIANDRA NALLAKKA GARI SUPRITHA\n";

    return 0;
}
```

Q4)

Java

```
/*
* WeatherTrack - Daily Weather Statistics Application
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
* MVVM Architecture with Room Database and WorkManager
* Complete Android weather tracking solution
*/

// ================================================================================
// 1. DATA LAYER - Entity and Database
// ================================================================================

// WeatherRecord.java
package com.supritha.weathertrack.data.entity;

import androidx.room.Entity;
import androidx.room.PrimaryKey;
import java.util.Date;

/**
* Weather record entity for Room database
```

```
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
@Entity(tableName = "weather_records")
public class WeatherRecord {
   @PrimaryKey(autoGenerate = true)
   private int recordId;

   private double temperature;
   private double humidity;
   private String weatherCondition;
   private String cityName;
   private Date recordTimestamp;
   private long fetchTimestamp;

   // Constructor
   public WeatherRecord(double temperature, double humidity, String
weatherCondition,
                        String cityName, Date recordTimestamp) {
       this.temperature = temperature;
       this.humidity = humidity;
       this.weatherCondition = weatherCondition;
       this.cityName = cityName;
       this.recordTimestamp = recordTimestamp;
       this.fetchTimestamp = System.currentTimeMillis();
   }

   // Getters and Setters
   public int getRecordId() { return recordId; }
   public void setRecordId(int recordId) { this.recordId = recordId; }

   public double getTemperature() { return temperature; }
   public void setTemperature(double temperature) { this.temperature =
temperature; }

   public double getHumidity() { return humidity; }
   public void setHumidity(double humidity) { this.humidity = humidity; }

   public String getWeatherCondition() { return weatherCondition; }
   public void setWeatherCondition(String weatherCondition) {
this.weatherCondition = weatherCondition; }

   public String getCityName() { return cityName; }
   public void setCityName(String cityName) { this.cityName = cityName; }
```

```
    public Date getRecordTimestamp() { return recordTimestamp; }
    public void setRecordTimestamp(Date recordTimestamp) { this.recordTimestamp
= recordTimestamp; }

    public long getFetchTimestamp() { return fetchTimestamp; }
    public void setFetchTimestamp(long fetchTimestamp) { this.fetchTimestamp =
fetchTimestamp; }
}

// WeatherDao.java
package com.supritha.weathertrack.data.dao;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;
import com.supritha.weathertrack.data.entity.WeatherRecord;
import java.util.Date;
import java.util.List;

/**
* Data Access Object for weather records
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
@Dao
public interface WeatherDao {

    @Insert
    void insertWeatherRecord(WeatherRecord weatherRecord);

    @Query("SELECT * FROM weather_records ORDER BY recordTimestamp DESC")
    LiveData<List<WeatherRecord>> getAllWeatherRecords();

    @Query("SELECT * FROM weather_records WHERE recordTimestamp >= :startDate
ORDER BY recordTimestamp DESC")
    LiveData<List<WeatherRecord>> getWeatherRecordsFromDate(Date startDate);

    @Query("SELECT * FROM weather_records WHERE recordTimestamp >= :startDate
AND recordTimestamp <= :endDate ORDER BY recordTimestamp DESC")
    List<WeatherRecord> getWeatherRecordsBetweenDates(Date startDate, Date
endDate);

    @Query("SELECT * FROM weather_records ORDER BY recordTimestamp DESC LIMIT
1")
```

```
    WeatherRecord getLatestWeatherRecord();

    @Query("SELECT COUNT(*) FROM weather_records")
    int getRecordCount();

    @Query("DELETE FROM weather_records WHERE recordTimestamp < :cutoffDate")
    void deleteOldRecords(Date cutoffDate);
}

// WeatherDatabase.java
package com.supritha.weathertrack.data.database;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import androidx.room.TypeConverters;
import android.content.Context;
import com.supritha.weathertrack.data.dao.WeatherDao;
import com.supritha.weathertrack.data.entity.WeatherRecord;
import com.supritha.weathertrack.data.converter.DateConverter;

/**
* Room database configuration
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
@Database(
    entities = {WeatherRecord.class},
    version = 1,
    exportSchema = false
)
@TypeConverters({DateConverter.class})
public abstract class WeatherDatabase extends RoomDatabase {

    private static volatile WeatherDatabase DATABASE_INSTANCE;
    private static final String DATABASE_NAME = "weather_track_database";

    public abstract WeatherDao weatherDao();

    public static WeatherDatabase getInstance(Context context) {
        if (DATABASE_INSTANCE == null) {
            synchronized (WeatherDatabase.class) {
                if (DATABASE_INSTANCE == null) {
                    DATABASE_INSTANCE = Room.databaseBuilder(
                        context.getApplicationContext(),
```

```
                    WeatherDatabase.class,
                    DATABASE_NAME
                ).build();
            }
        }
    }
    return DATABASE_INSTANCE;
  }
}

// DateConverter.java
package com.supritha.weathertrack.data.converter;

import androidx.room.TypeConverter;
import java.util.Date;

/**
* Type converter for Date objects in Room
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class DateConverter {

  @TypeConverter
  public static Date fromTimestamp(Long value) {
      return value == null ? null : new Date(value);
  }

  @TypeConverter
  public static Long dateToTimestamp(Date date) {
      return date == null ? null : date.getTime();
  }
}

// ============================================================================
// 2. API LAYER - Mock Weather Service
// ============================================================================

// WeatherApiService.java
package com.supritha.weathertrack.data.api;

import com.supritha.weathertrack.data.model.WeatherResponse;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;
```

```java
/**
* Weather API service interface
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public interface WeatherApiService {

   @GET("weather/current")
   Call<WeatherResponse> getCurrentWeather(@Query("city") String cityName);
}

// WeatherResponse.java
package com.supritha.weathertrack.data.model;

import com.google.gson.annotations.SerializedName;

/**
* Weather API response model
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class WeatherResponse {

   @SerializedName("temperature")
   private double temperature;

   @SerializedName("humidity")
   private double humidity;

   @SerializedName("condition")
   private String weatherCondition;

   @SerializedName("city")
   private String cityName;

   @SerializedName("timestamp")
   private long timestamp;

   @SerializedName("success")
   private boolean success;

   @SerializedName("error_message")
   private String errorMessage;

   // Constructors
```

```
    public WeatherResponse() {}

    public WeatherResponse(double temperature, double humidity, String
weatherCondition, String cityName) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.weatherCondition = weatherCondition;
        this.cityName = cityName;
        this.timestamp = System.currentTimeMillis();
        this.success = true;
    }

    // Getters and Setters
    public double getTemperature() { return temperature; }
    public void setTemperature(double temperature) { this.temperature =
temperature; }

    public double getHumidity() { return humidity; }
    public void setHumidity(double humidity) { this.humidity = humidity; }

    public String getWeatherCondition() { return weatherCondition; }
    public void setWeatherCondition(String weatherCondition) {
this.weatherCondition = weatherCondition; }

    public String getCityName() { return cityName; }
    public void setCityName(String cityName) { this.cityName = cityName; }

    public long getTimestamp() { return timestamp; }
    public void setTimestamp(long timestamp) { this.timestamp = timestamp; }

    public boolean isSuccess() { return success; }
    public void setSuccess(boolean success) { this.success = success; }

    public String getErrorMessage() { return errorMessage; }
    public void setErrorMessage(String errorMessage) { this.errorMessage =
errorMessage; }
}

// MockWeatherService.java
package com.supritha.weathertrack.data.service;

import com.supritha.weathertrack.data.model.WeatherResponse;
import java.util.Random;
```

```java
/**
* Mock weather service for demonstration
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class MockWeatherService {

   private static final String[] WEATHER_CONDITIONS = {
       "Sunny", "Cloudy", "Rainy", "Partly Cloudy", "Thunderstorm", "Foggy",
"Windy"
   };

   private static final String[] SAMPLE_CITIES = {
       "New York", "London", "Tokyo", "Sydney", "Mumbai", "Berlin", "Paris"
   };

   private Random randomGenerator;

   public MockWeatherService() {
       this.randomGenerator = new Random();
   }

   public WeatherResponse fetchCurrentWeather(String cityName) {
       // Simulate network delay
       try {
           Thread.sleep(1000 + randomGenerator.nextInt(2000));
       } catch (InterruptedException e) {
           Thread.currentThread().interrupt();
       }

       // Simulate occasional API failures
       if (randomGenerator.nextDouble() < 0.1) { // 10% failure rate
           WeatherResponse errorResponse = new WeatherResponse();
           errorResponse.setSuccess(false);
           errorResponse.setErrorMessage("Weather service temporarily
unavailable");
           return errorResponse;
       }

       // Generate realistic mock data
       double baseTemperature = getBaseTemperatureForCity(cityName);
       double temperatureVariation = (randomGenerator.nextDouble() - 0.5) * 10;
       double currentTemperature = baseTemperature + temperatureVariation;

       double humidity = 30 + randomGenerator.nextDouble() * 40; // 30-70%
```

```
        String condition =
WEATHER_CONDITIONS[randomGenerator.nextInt(WEATHER_CONDITIONS.length)];

        return new WeatherResponse(currentTemperature, humidity, condition,
cityName);
    }

    private double getBaseTemperatureForCity(String cityName) {
        // Return different base temperatures for different cities
        switch (cityName.toLowerCase()) {
            case "mumbai": return 28.0;
            case "new york": return 15.0;
            case "london": return 12.0;
            case "tokyo": return 18.0;
            case "sydney": return 22.0;
            case "berlin": return 10.0;
            case "paris": return 14.0;
            default: return 20.0;
        }
    }
}

// ================================================================================
// 3. REPOSITORY LAYER
// ================================================================================

// WeatherRepository.java
package com.supritha.weathertrack.repository;

import android.content.Context;
import android.util.Log;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import com.supritha.weathertrack.data.database.WeatherDatabase;
import com.supritha.weathertrack.data.entity.WeatherRecord;
import com.supritha.weathertrack.data.model.WeatherResponse;
import com.supritha.weathertrack.data.service.MockWeatherService;
import com.supritha.weathertrack.utils.NetworkUtils;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

```java
/**
* Weather data repository - Single source of truth
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class WeatherRepository {

   private static final String TAG = "WeatherRepository";
   private static volatile WeatherRepository REPOSITORY_INSTANCE;

   private WeatherDatabase weatherDatabase;
   private MockWeatherService mockWeatherService;
   private ExecutorService databaseExecutor;
   private Context applicationContext;

   // LiveData for UI observation
   private MutableLiveData<String> errorMessageLiveData;
   private MutableLiveData<Boolean> loadingStateLiveData;

   private WeatherRepository(Context context) {
       this.applicationContext = context.getApplicationContext();
       this.weatherDatabase = WeatherDatabase.getInstance(applicationContext);
       this.mockWeatherService = new MockWeatherService();
       this.databaseExecutor = Executors.newFixedThreadPool(4);
       this.errorMessageLiveData = new MutableLiveData<>();
       this.loadingStateLiveData = new MutableLiveData<>();

       Log.d(TAG, "WeatherRepository initialized by ADIANDRA NALLAKKA GARI
SUPRITHA");
   }

   public static WeatherRepository getInstance(Context context) {
       if (REPOSITORY_INSTANCE == null) {
           synchronized (WeatherRepository.class) {
               if (REPOSITORY_INSTANCE == null) {
                   REPOSITORY_INSTANCE = new WeatherRepository(context);
               }
           }
       }
       return REPOSITORY_INSTANCE;
   }

   // Fetch and save weather data
   public void fetchAndSaveWeatherData(String cityName) {
       loadingStateLiveData.postValue(true);
```

```
        databaseExecutor.execute(() -> {
            try {
                // Check network connectivity
                if (!NetworkUtils.isNetworkAvailable(applicationContext)) {
                    errorMessageLiveData.postValue("No internet connection.
Please check your network.");
                    loadingStateLiveData.postValue(false);
                    return;
                }

                // Fetch weather data from mock service
                WeatherResponse weatherResponse =
mockWeatherService.fetchCurrentWeather(cityName);

                if (weatherResponse.isSuccess()) {
                    // Convert to database entity and save
                    WeatherRecord newRecord = new WeatherRecord(
                        weatherResponse.getTemperature(),
                        weatherResponse.getHumidity(),
                        weatherResponse.getWeatherCondition(),
                        weatherResponse.getCityName(),
                        new Date()
                    );

                    weatherDatabase.weatherDao().insertWeatherRecord(newRecord);

                    Log.d(TAG, "Weather data saved successfully for " +
cityName);
                    errorMessageLiveData.postValue(null); // Clear any previous
errors

                } else {
                    errorMessageLiveData.postValue("Weather service error: " +
weatherResponse.getErrorMessage());
                }

            } catch (Exception e) {
                Log.e(TAG, "Error fetching weather data", e);
                errorMessageLiveData.postValue("Failed to fetch weather data.
Please try again.");
            } finally {
                loadingStateLiveData.postValue(false);
            }
```

```
        });
    }

    // Get all weather records
    public LiveData<List<WeatherRecord>> getAllWeatherRecords() {
        return weatherDatabase.weatherDao().getAllWeatherRecords();
    }

    // Get weather records for the past 7 days
    public LiveData<List<WeatherRecord>> getWeeklyWeatherRecords() {
        Calendar calendar = Calendar.getInstance();
        calendar.add(Calendar.DAY_OF_YEAR, -7);
        Date sevenDaysAgo = calendar.getTime();

        return
weatherDatabase.weatherDao().getWeatherRecordsFromDate(sevenDaysAgo);
    }

    // Get weather records for a specific date range
    public void getWeatherRecordsForDateRange(Date startDate, Date endDate,

RepositoryCallback<List<WeatherRecord>> callback) {
        databaseExecutor.execute(() -> {
            try {
                List<WeatherRecord> records = weatherDatabase.weatherDao()
                    .getWeatherRecordsBetweenDates(startDate, endDate);
                callback.onSuccess(records);
            } catch (Exception e) {
                Log.e(TAG, "Error fetching date range records", e);
                callback.onError("Database error: " + e.getMessage());
            }
        });
    }

    // Clean up old records (keep only last 30 days)
    public void cleanupOldRecords() {
        databaseExecutor.execute(() -> {
            try {
                Calendar calendar = Calendar.getInstance();
                calendar.add(Calendar.DAY_OF_YEAR, -30);
                Date thirtyDaysAgo = calendar.getTime();

                weatherDatabase.weatherDao().deleteOldRecords(thirtyDaysAgo);
                Log.d(TAG, "Old weather records cleaned up successfully");
```

```
            } catch (Exception e) {
                Log.e(TAG, "Error cleaning up old records", e);
            }
        });
    }

    // LiveData getters for UI observation
    public LiveData<String> getErrorMessage() {
        return errorMessageLiveData;
    }

    public LiveData<Boolean> getLoadingState() {
        return loadingStateLiveData;
    }

    // Repository callback interface
    public interface RepositoryCallback<T> {
        void onSuccess(T data);
        void onError(String errorMessage);
    }
}

// ================================================================================
// 4. WORKMANAGER - Background Sync
// ================================================================================

// WeatherSyncWorker.java
package com.supritha.weathertrack.worker;

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Log;
import androidx.annotation.NonNull;
import androidx.work.Worker;
import androidx.work.WorkerParameters;
import com.supritha.weathertrack.repository.WeatherRepository;

/**
* Background worker for automatic weather data sync
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class WeatherSyncWorker extends Worker {

    private static final String TAG = "WeatherSyncWorker";
```

```
    private static final String PREFS_NAME = "WeatherTrackPrefs";
    private static final String PREF_DEFAULT_CITY = "default_city";

    public WeatherSyncWorker(@NonNull Context context, @NonNull WorkerParameters
workerParams) {
        super(context, workerParams);
    }

    @NonNull
    @Override
    public Result doWork() {
        Log.d(TAG, "Starting background weather sync - by ADIANDRA NALLAKKA GARI
SUPRITHA");

        try {
            // Get user's preferred city from SharedPreferences
            SharedPreferences preferences = getApplicationContext()
                .getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
            String defaultCity = preferences.getString(PREF_DEFAULT_CITY, "New
York");

            // Initialize repository and fetch weather data
            WeatherRepository repository =
WeatherRepository.getInstance(getApplicationContext());

            // This will run synchronously in the worker thread
            repository.fetchAndSaveWeatherData(defaultCity);

            // Also cleanup old records during background sync
            repository.cleanupOldRecords();

            Log.d(TAG, "Background weather sync completed successfully for " +
defaultCity);
            return Result.success();

        } catch (Exception e) {
            Log.e(TAG, "Background weather sync failed", e);
            return Result.retry(); // Retry the work
        }
    }
}

// WorkManagerScheduler.java
package com.supritha.weathertrack.utils;
```

```java
import android.content.Context;
import androidx.work.Constraints;
import androidx.work.ExistingPeriodicWorkPolicy;
import androidx.work.NetworkType;
import androidx.work.PeriodicWorkRequest;
import androidx.work.WorkManager;
import com.supritha.weathertrack.worker.WeatherSyncWorker;
import java.util.concurrent.TimeUnit;

/**
* WorkManager scheduling utility
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class WorkManagerScheduler {

   private static final String WEATHER_SYNC_WORK_NAME =
"weather_sync_periodic_work";

   public static void schedulePeriodicWeatherSync(Context context) {
       // Define constraints for the work
       Constraints syncConstraints = new Constraints.Builder()
           .setRequiredNetworkType(NetworkType.CONNECTED)
           .setRequiresBatteryNotLow(true)
           .build();

       // Create periodic work request (every 6 hours)
       PeriodicWorkRequest weatherSyncRequest = new
PeriodicWorkRequest.Builder(
           WeatherSyncWorker.class,
           6, TimeUnit.HOURS,  // Repeat interval
           1, TimeUnit.HOURS   // Flex interval
       )
       .setConstraints(syncConstraints)
       .addTag("weather_background_sync")
       .build();

       // Schedule the work
       WorkManager.getInstance(context).enqueueUniquePeriodicWork(
           WEATHER_SYNC_WORK_NAME,
           ExistingPeriodicWorkPolicy.KEEP, // Keep existing work if already
scheduled
           weatherSyncRequest
       );
```

```
    }

    public static void cancelPeriodicWeatherSync(Context context) {

WorkManager.getInstance(context).cancelUniqueWork(WEATHER_SYNC_WORK_NAME);
    }
}

// ================================================================================
// 5. VIEWMODEL LAYER
// ================================================================================

// WeatherViewModel.java
package com.supritha.weathertrack.viewmodel;

import android.app.Application;
import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import com.supritha.weathertrack.data.entity.WeatherRecord;
import com.supritha.weathertrack.repository.WeatherRepository;
import java.util.List;

/**
* ViewModel for weather data management
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class WeatherViewModel extends AndroidViewModel {

    private WeatherRepository weatherRepository;
    private LiveData<List<WeatherRecord>> allWeatherRecords;
    private LiveData<List<WeatherRecord>> weeklyWeatherRecords;
    private MutableLiveData<WeatherRecord> selectedWeatherRecord;

    public WeatherViewModel(@NonNull Application application) {
        super(application);
        weatherRepository = WeatherRepository.getInstance(application);
        allWeatherRecords = weatherRepository.getAllWeatherRecords();
        weeklyWeatherRecords = weatherRepository.getWeeklyWeatherRecords();
        selectedWeatherRecord = new MutableLiveData<>();
    }

    // Fetch fresh weather data
```

```
    public void refreshWeatherData(String cityName) {
        weatherRepository.fetchAndSaveWeatherData(cityName);
    }

    // LiveData getters for UI observation
    public LiveData<List<WeatherRecord>> getAllWeatherRecords() {
        return allWeatherRecords;
    }

    public LiveData<List<WeatherRecord>> getWeeklyWeatherRecords() {
        return weeklyWeatherRecords;
    }

    public LiveData<String> getErrorMessage() {
        return weatherRepository.getErrorMessage();
    }

    public LiveData<Boolean> getLoadingState() {
        return weatherRepository.getLoadingState();
    }

    public LiveData<WeatherRecord> getSelectedWeatherRecord() {
        return selectedWeatherRecord;
    }

    // Set selected weather record for detail view
    public void selectWeatherRecord(WeatherRecord record) {
        selectedWeatherRecord.setValue(record);
    }

    // Cleanup old records
    public void cleanupOldData() {
        weatherRepository.cleanupOldRecords();
    }
}

// ================================================================================
// 6. UTILITY CLASSES
// ================================================================================

// NetworkUtils.java
package com.supritha.weathertrack.utils;

import android.content.Context;
```

```java
import android.net.ConnectivityManager;
import android.net.NetworkInfo;

/**
* Network connectivity utility
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class NetworkUtils {

   public static boolean isNetworkAvailable(Context context) {
       ConnectivityManager connectivityManager =
           (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);

       if (connectivityManager != null) {
           NetworkInfo activeNetworkInfo =
connectivityManager.getActiveNetworkInfo();
           return activeNetworkInfo != null && activeNetworkInfo.isConnected();
       }

       return false;
   }
}

// DateUtils.java
package com.supritha.weathertrack.utils;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
* Date formatting utilities
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class DateUtils {

   private static final SimpleDateFormat DATE_FORMAT =
       new SimpleDateFormat("MMM dd, yyyy", Locale.getDefault());

   private static final SimpleDateFormat TIME_FORMAT =
       new SimpleDateFormat("HH:mm", Locale.getDefault());

   private static final SimpleDateFormat FULL_FORMAT =
```

```
        new SimpleDateFormat("MMM dd, yyyy HH:mm", Locale.getDefault());

    public static String formatDate(Date date) {
        return DATE_FORMAT.format(date);
    }

    public static String formatTime(Date date) {
        return TIME_FORMAT.format(date);
    }

    public static String formatDateTime(Date date) {
        return FULL_FORMAT.format(date);
    }
}

// ============================================================================
// 7. MAIN ACTIVITY
// ============================================================================

// MainActivity.java
package com.supritha.weathertrack;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.supritha.weathertrack.adapter.WeatherRecordAdapter;
import com.supritha.weathertrack.utils.WorkManagerScheduler;
import com.supritha.weathertrack.viewmodel.WeatherViewModel;

/**
* Main Activity for WeatherTrack App
* Author: ADIANDRA NALLAKKA GARI SUPRITHA
*/
public class MainActivity extends AppCompatActivity {

    private WeatherViewModel weatherViewModel;
    private WeatherRecordAdapter weatherAdapter;
```

```
    private TextView titleTextView;
    private Button refreshButton;
    private ProgressBar loadingProgressBar;
    private RecyclerView weatherRecyclerView;
    private TextView errorTextView;
    private TextView emptyStateTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initialize ViewModel
        weatherViewModel = new
ViewModelProvider(this).get(WeatherViewModel.class);

        // Initialize UI components
        initializeViews();
        setupRecyclerView();
        setupClickListeners();
        observeViewModel();

        // Schedule background work
        WorkManagerScheduler.schedulePeriodicWeatherSync(this);

        // Initial weather fetch
        weatherViewModel.refreshWeatherData("New York");

        // Display author info
        Toast.makeText(this, "WeatherTrack by ADIANDRA NALLAKKA GARI SUPRITHA",
                       Toast.LENGTH_LONG).show();
    }

    private void initializeViews() {
        titleTextView = findViewById(R.id.titleTextView);
        refreshButton = findViewById(R.id.refreshButton);
        loadingProgressBar = findViewById(R.id.loadingProgressBar);
        weatherRecyclerView = findViewById(R.id.weatherRecyclerView);
        errorTextView = findViewById(R.id.errorTextView);
        emptyStateTextView = findViewById(R.id.emptyStateTextView);

        titleTextView.setText("WeatherTrack - Daily Weather Stats");
    }
```

```
    private void setupRecyclerView() {
        weatherAdapter = new WeatherRecordAdapter();
        weatherRecyclerView.setLayoutManager(new LinearLayoutManager(this));
        weatherRecyclerView.setAdapter(weatherAdapter);

        // Handle item clicks for detail view
        weatherAdapter.setOnItemClickListener(record -> {
            weatherViewModel.selectWeatherRecord(record);
            // Here you would typically navigate to a detail activity
            showWeatherDetail(record);
        });
    }

    private void setupClickListeners() {
        refreshButton.setOnClickListener(v -> {
            weatherViewModel.refreshWeatherData("New York");
        });
    }

    private void observeViewModel() {
        // Observe weather records
        weatherViewModel.getWeeklyWeatherRecords().observe(this, records -> {
            if (records != null && !records.isEmpty()) {
                weatherAdapter.updateWeatherRecords(records);
                weatherRecyclerView.setVisibility(View.VISIBLE);
                emptyStateTextView.setVisibility(View.GONE);
            } else {
                weatherRecyclerView.setVisibility(View.GONE);
                emptyStateTextView.setVisibility(View.VISIBLE);
                emptyStateTextView.setText("No weather data available. Pull to
refresh!");
            }
        });

        // Observe loading state
        weatherViewModel.getLoadingState().observe(this, isLoading -> {
            if (isLoading != null) {
                loadingProgressBar.setVisibility(isLoading ? View.VISIBLE :
View.GONE);
                refreshButton.setEnabled(!isLoading);
            }
        });
```

```
        // Observe error messages
        weatherViewModel.getErrorMessage().observe(this, errorMessage -> {
            if (errorMessage != null && !errorMessage.isEmpty()) {
                errorTextView.setText(errorMessage);
                errorTextView.setVisibility(View.VISIBLE);
                Toast.makeText(this, errorMessage, Toast.LENGTH_LONG).show();
            } else {
                errorTextView.setVisibility(View.GONE);
            }
        });
    }

    private void showWeatherDetail(WeatherRecord record) {
        String detailMessage = String.format(
            "Weather Details\n\n" +
            "Temperature: %.1f°C\n" +
            "Humidity: %.1f%%\n" +
            "Condition: %s\n" +
            "City: %s\n" +
            "Recorded: %s",
            record.getTemperature(),
            record.getHumidity(),
            record.getWeatherCondition(),
            record.getCityName(),
            record.getRecordTimestamp().toString()
        );

        Toast.makeText(this, detailMessage, Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Cleanup would go here if needed
    }
}
```