# Proyecto 2: Plataforma CodeCoach

Antony Javier Hernández Castillo
Carnet: 2022321746
*Escuela de Ingeniería en Computadores*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica

Dylan Maximiliano Guerrero González
Carnet: 2022016016
*Escuela de Ingeniería en Computadores*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica

José Fabio Ruiz Morales
Carnet: 2023138210
*Escuela de Ingeniería en Computadores*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica

## Índice



Curso: CE2103 — Algoritmos y Estructuras de Datos II.
Instituto Tecnológico de Costa Rica.
Escuela de Ingeniería en Computadores.
Profesor: Ing. Isaac Ramírez, M.SI.

## I. Introducción

CodeCoach es una plataforma de preparación y práctica de ejercicios de ciencias de computación diseñada para entrevistas laborales en empresas de software. El proyecto surge como respuesta a la necesidad de los estudiantes y profesionales de tecnología de prepararse para los retos de programación que comúnmente se presentan en procesos de selección de empresas como Google, Amazon, Meta y Microsoft, siguiendo el modelo de plataformas reconocidas como LeetCode y HackerRank.

La plataforma implementa una arquitectura distribuida basada en microservicios que permite a los usuarios seleccionar problemas de programación, escribir soluciones en C++, ejecutarlas en un ambiente aislado (sandbox), y recibir retroalimentación inteligente sobre la calidad algorítmica de sus soluciones. El flujo funcional de la plataforma se describe a continuación:

1. El usuario selecciona un problema en la interfaz gráfica.
2. El Gestor de Problemas devuelve el enunciado y ejemplos desde MongoDB.
3. El usuario envía su solución en código C++.
4. El Motor de Evaluación ejecuta el código en un sandbox seguro (Docker):
   - Compila el código con g++.
   - Ejecuta pruebas predefinidas.
   - Mide tiempo y memoria.
   - Envía resultados.
5. El Analizador de Soluciones estima la complejidad algorítmica y reconoce el tipo de algoritmo usado.
6. El LLM Coach (Gemini API) recibe el código, resultados y análisis algorítmico, respondiendo con feedback adaptado (pistas, explicación de error, sugerencias de optimización).
7. El usuario recibe el estado de los tests, complejidad estimada y feedback del coach AI.

La plataforma resuelve problemas reales como la falta de retroalimentación inmediata en el aprendizaje de algoritmos, la dificultad para identificar patrones algorítmicos, y la necesidad de practicar en un ambiente similar al de entrevistas técnicas reales.

## II. Descripción de la Solución

Esta sección presenta la implementación detallada de cada uno de los cuatro componentes principales del sistema: In-

terfaz Gráfica, Gestor de Problemas, Motor de Evaluación y Analizador de Soluciones.

### *II-A. Interfaz Gráfica*

*II-A1. Descripción del Requerimiento: "Se recomienda que los grupos de trabajo se familiaricen con LeetCode y Hackerrank para darse una idea de los elementos básicos con los que debe contar la interfaz gráfica de una plataforma como la solicitada en este proyecto. Queda a criterio del grupo el lenguaje de programación y tecnologías que utilicen para implementar dicha interfaz."*

*II-A2. Implementación del Requerimiento:* La interfaz gráfica fue desarrollada en C# utilizando Windows Presentation Foundation (WPF) con el framework .NET 8.0. Se optó por esta tecnología debido a su robusta capacidad de crear interfaces modernas y responsivas, además de su excelente integración con servicios HTTP para comunicarse con los backends en C++.

**Estructuras de datos utilizadas:**

- `ProblemDto`: Clase que representa un problema completo con todas sus propiedades (id, título, descripción, dificultad, tags, casos de prueba y código inicial).
- `TestCaseDto`: Estructura para almacenar pares input/expected_output.
- `EvaluationResponse`: DTO para deserializar respuestas del Motor de Evaluación.
- `AnalysisResult`: Clase que contiene el resultado del análisis de complejidad.
- `List<ProblemDto>`: Colección dinámica para manejar la lista de problemas filtrados.

**Arquitectura de la UI:**

La aplicación sigue el patrón de separación de responsabilidades con tres capas principales:

Listing 1. Estructura de carpetas de la UI

```
UI/
        MainWindow.xaml(.cs)       // Ventana
    principal
        AdminWindow.xaml(.cs)      // Administracin
        Models/
            ProblemDto.cs          // DTOs de
    problemas
            EvaluationDtos.cs      // DTOs de
    evaluacin
        Services/
            ApiClient.cs           // Cliente HTTP
    base
            ProblemApiClient.cs    // Cliente del
    Gestor
            AnalyzerApiClient.cs   // Cliente del
    Analizador
        UI.csproj
```

**Componentes principales de MainWindow:**

Listing 2. Inicialización de clientes API

```
public partial class MainWindow : Window
{
    private readonly ProblemApiClient _apiClient;
    private readonly AnalyzerApiClient
    _analyzerService;
    private ProblemDto? _currentProblem;
    private bool _serverConnected;

    public MainWindow()
    {
        InitializeComponent();
        // Cliente para Gestor (puerto 8080)
        _apiClient = new ProblemApiClient(
            "http://localhost:8080");
        // Cliente para Analizador (puerto 8081)
        _analyzerService = new AnalyzerApiClient(
            "http://localhost:8081");

        Loaded += MainWindow_Loaded;
    }
}
```

**Sistema de filtrado:**

Se implementó un sistema de filtrado por dificultad y tags que permite a los usuarios encontrar problemas específicos:

Listing 3. Filtrado por dificultad

```
private async void
    DifficultyComboBox_SelectionChanged(
    object sender, SelectionChangedEventArgs e)
{
    var selected = DifficultyComboBox.SelectedItem
        as ComboBoxItem;
    if (selected?.Content?.ToString() == "Todas")
    {
        await LoadAllProblemsAsync();
    }
    else
    {
        var difficulty = selected?.Content?.ToString
    ();
        var filtered = await _apiClient
            .GetProblemsByDifficultyAsync(difficulty
    );
        UpdateProblemList(filtered);
    }
}
```

**Editor de código con resaltado de sintaxis:**

Se integró AvalonEdit para proporcionar un editor de código profesional con resaltado de sintaxis para C++:

Listing 4. Configuración del editor en XAML

```
<avalonedit:TextEditor x:Name="CodeEditor"
    FontFamily="Consolas"
    FontSize="14"
    Background="#1e1e1e"
    Foreground="White"
    ShowLineNumbers="True"
    VerticalScrollBarVisibility="Auto"
    HorizontalScrollBarVisibility="Auto" />
```

**Alternativas consideradas:**

Se evaluaron otras tecnologías como Electron (JavaScript) y Qt (C++), pero WPF fue seleccionado por su mejor integración con el ecosistema .NET y su capacidad de crear interfaces nativas de Windows con alto rendimiento.

**Problemas encontrados:**

- La comunicación asíncrona con múltiples backends requirió manejo cuidadoso de excepciones y timeouts.
- El resaltado de sintaxis personalizado para C++ requirió crear un archivo de definición XSHD.

*II-A3. Interacción en la UI:* **Ventana Principal (MainWindow):**

1. El usuario ve la lista de problemas disponibles en el panel izquierdo.
2. Puede filtrar por dificultad usando el ComboBox superior.
3. Al seleccionar un problema, el panel derecho muestra la descripción completa.
4. El editor de código en la parte central permite escribir la solución.
5. El botón “Run” envía el código al Motor de Evaluación.
6. El botón “Submit” envía el código al Analizador para obtener feedback.
7. La consola inferior muestra los resultados en tiempo real.

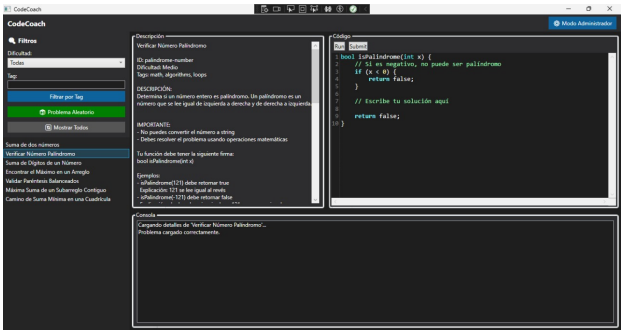


Figura 1. Ventana principal de CodeCoach mostrando la lista de problemas, editor de código y consola de resultados

### Ventana de Administración (AdminWindow):

1. El administrador accede mediante el botón “Modo Administrador”.
2. El panel izquierdo muestra todos los problemas existentes.
3. El panel derecho contiene el formulario de edición completo.
4. Se pueden agregar/eliminar casos de prueba individualmente.
5. El botón “Guardar” persiste los cambios en MongoDB.

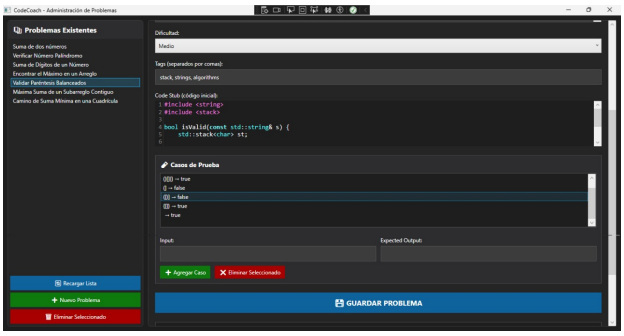


Figura 2. Ventana de administración para gestionar problemas de programación

## *II-B. Gestor de Problemas*

*II-B1. Descripción del Requerimiento: “La interfaz gráfica provee un modo de administración que permite ingresar problemas de código en el formato que el equipo de trabajo determine. Dicha interfaz se comunica con un REST API en C++. Dicho API adapta los datos recibidos desde la interfaz y los almacena en una base de datos MongoDB. El gestor de problemas también recibe peticiones de la interfaz de usuario cuando se solicite un nuevo problema de cierta categoría solicitada por el usuario.”*

*II-B2. Implementación del Requerimiento:* El Gestor de Problemas fue implementado como un servidor REST en C++ utilizando el framework Crow para el manejo de rutas HTTP y el driver oficial mongocxx para la conexión con MongoDB.

**Estructuras de datos utilizadas:**

Listing 5. Estructuras de datos del Gestor

```
// Caso de prueba individual
struct TestCase {
    std::string input;
    std::string expected_output;
};

// Problema completo
struct Problem {
    std::string problem_id;    // Primary key
    std::string title;
    std::string description;
    std::string difficulty;    // Facil, Medio, Dificil
    std::vector<std::string> tags;
    std::vector<TestCase> test_cases;
    std::string code_stub;     // Codigo inicial
};
```

**Arquitectura del repositorio:**

Se implementó el patrón Repository para encapsular toda la lógica de acceso a datos:

Listing 6. Interfaz del ProblemRepository

```
class ProblemRepository {
public:
    explicit ProblemRepository(mongocxx::database db
    );

    void insert_problem(const Problem& p);
    std::vector<Problem> get_all();
    std::optional<Problem> get_by_id(
        const std::string& id);
    std::vector<Problem> get_by_difficulty(
        const std::string& difficulty);
    bool update_problem(const Problem& p);
    bool delete_problem(const std::string& id);

private:
    mongocxx::collection collection_;
};
```

**API REST disponible:**

Cuadro I
ENDPOINTS DEL GESTOR DE PROBLEMAS

| Método | Ruta | Descripción |
|---|---|---|
| GET | /health | Health check |
| GET | /problems | Lista todos los problemas |
| GET | /problems/<id> | Obtiene un problema |
| GET | /problems/random | Problema aleatorio |
| GET | /problems/difficulty/<level> | Filtrar por dificultad |
| GET | /problems/tags/<tag> | Filtrar por tag |
| POST | /problems | Crear problema |
| PUT | /problems/<id> | Actualizar problema |
| DELETE | /problems/<id> | Eliminar problema |
| POST | /submissions | Evaluar solución |
| POST | /run | Ejecutar código simple |

**Conversión BSON/JSON:**

Una parte crítica del Gestor es la conversión entre documentos BSON de MongoDB y estructuras C++:

Listing 7. Conversión de documento BSON a Problem

```
static Problem document_to_problem(
    const bsoncxx::document::view& doc_view)
{
    Problem p;
    p.problem_id = get_string_field(doc_view,
        "problem_id");
    p.title = get_string_field(doc_view, "title");
    p.description = get_string_field(doc_view,
        "description");
    p.difficulty = get_string_field(doc_view,
        "difficulty");
    p.code_stub = get_string_field(doc_view,
        "code_stub");

    // Parsear array de tags
    auto it_tags = doc_view.find("tags");
    if (it_tags != doc_view.end()) {
        auto arr = it_tags->get_array().value;
        for (auto&& t : arr) {
            p.tags.emplace_back(
                t.get_string().value.data());
        }
    }

    // Parsear test_cases...
    return p;
}
```

**Proxy hacia el Motor de Evaluación:**

El Gestor actúa como proxy para las solicitudes de evaluación, enriqueciendo la petición con los casos de prueba almacenados:

Listing 8. Endpoint de submissions

```
CROW_ROUTE(app, "/submissions")
    .methods(crow::HTTPMethod::Post)
([&repo](const crow::request& req) {
    // Parsear peticion
    auto body = crow::json::load(req.body);
    std::string problem_id = body["problem_id"].s();
    std::string source_code = body["source_code"].s
    ();

    // Obtener problema con test_cases
    auto maybe_problem = repo.get_by_id(problem_id);
    if (!maybe_problem) {
        return make_error_response(404,
            "Problema no encontrado");
    }

    // Construir JSON para el Motor
    crow::json::wvalue eval_json;
    eval_json["submission_id"] = "sub-" + problem_id
    ;
    eval_json["source_code"] = source_code;
    // Agregar test_cases...

    // Llamar al Motor via HTTP
    cpr::Response resp = cpr::Post(
        cpr::Url{"http://localhost:8090/evaluate"},
        cpr::Header{{"Content-Type",
            "application/json"}},
        cpr::Body{eval_json.dump()}
    );

    // Reenviar respuesta
    return crow::response(200, resp.text);
});
```

**Alternativas consideradas:**

Se evaluó usar PostgreSQL como base de datos, pero MongoDB fue preferido por su flexibilidad para almacenar documentos JSON con estructura variable (diferentes cantidades de test_cases por problema) y su excelente integración con C++ mediante mongocxx.

**Limitaciones conocidas:**

- La conexión a MongoDB es local (localhost:27017).

*II-B3. Interacción en la UI:*

1. En AdminWindow, el usuario completa el formulario con los datos del problema.
2. Al presionar "Guardar", la UI serializa el ProblemDto a JSON.
3. Se envía una petición POST/PUT a `/problems`.
4. El Gestor valida los campos obligatorios.
5. Si es válido, se inserta/actualiza en MongoDB.
6. La respuesta JSON confirma el éxito o reporta errores.

### *II-C. Motor de Evaluación*

*II-C1. Descripción del Requerimiento: "El equipo de trabajo debe diseñar e implementar un intérprete de C++ para ejecutar el código ingresado por el usuario, medir el tiempo de ejecución y validar la solución contra las pruebas predefinidas que se debieron de ingresar cuando se registró el problema en el Gestor de Problemas. El motor de evaluación recibe las peticiones para ejecutar u obtener los resultados mediante un REST API. La ejecución del código debe realizarse en un ambiente aislado al del proceso en el que se ejecuta el motor de evaluación."*

*II-C2. Implementación del Requerimiento:* El Motor de Evaluación fue diseñado con una arquitectura modular que separa las responsabilidades en componentes especializados. La ejecución aislada se logra mediante contenedores Docker.

**Estructuras de datos utilizadas:**

Listing 9. Modelos del Motor de Evaluación

```
namespace engine {

// Solicitud de evaluacion
struct SubmissionRequest {
    std::string submissionId;
    std::string problemId;
    std::string language;
    std::string sourceCode;
    int timeLimitMs;
    int memoryLimitKb;
    std::vector<TestCase> testCases;
};

// Resultado de compilacion
struct CompileResult {
    int exitCode;
    std::string logFilePath;
};

// Limites de ejecucion
struct RunLimits {
    int timeLimitSeconds = 2;
    int memoryLimitMb = 256;
    double cpuLimit = 1.0;
```

```
    int pidsLimit = 64;
};

// Estado de un test
enum class TestStatus {
    Accepted,
    WrongAnswer,
    RuntimeError,
    TimeLimitExceeded,
    InternalError
};

// Resultado por test
struct TestResult {
    std::string testId;
    TestStatus status;
    int timeMs;
    int memoryKb;
    std::string runtimeLog;
};

// Resultado global
struct EvaluationResult {
    std::string submissionId;
    OverallStatus overallStatus;
    std::string compileLog;
    int maxTimeMs;
    int maxMemoryKb;
    std::vector<TestResult> tests;
};

}
```

**Componentes del Motor:**

1. **SubmissionFilesystem**: Maneja la creación de directorios y archivos para cada submission.
2. **DockerRunner**: Encapsula la interacción con Docker para compilar y ejecutar código.
3. **OutputComparer**: Compara la salida del programa con la salida esperada.
4. **EvaluationService**: Orquesta todo el flujo de evaluación.

**Ejecución aislada con Docker:**

Listing 10. Compilación en Docker

```
CompileResult DockerRunner::compile(
    const std::filesystem::path& submissionDir,
    const std::string& sourceFileName) const
{
    CompileResult result;
    std::string volumeArg =
        buildVolumeArgument(submissionDir);

    std::ostringstream cmd;
    cmd << "docker run --rm "
        << "--network=none "       // Sin red
        << "--memory=512m "        // Limite memoria
        << "--cpus=1 "             // Una CPU
        << volumeArg
        << imageName_ << " "
        << "/bin/bash -lc \"cd /workspace && "
        << "g++ " << sourceFileName
        << " -O2 -std=c++20 -o main "
        << "2> compile.log\"";

    result.logFilePath =
        (submissionDir / "compile.log").string();
    result.exitCode = std::system(cmd.str().c_str())
    ;

    return result;
}
```

**Ejecución de tests con límites:**

Listing 11. Ejecución de un test individual

```
RunResult DockerRunner::runSingleTest(
    const std::filesystem::path& submissionDir,
    const std::string& inputFileName,
    const std::string& outputFileName,
    const std::string& runtimeLogName,
    int timeLimitSeconds,
    const RunLimits& limits) const
{
    RunResult result;
    std::string volumeArg =
        buildVolumeArgument(submissionDir);

    std::ostringstream cmd;
    cmd << "docker run --rm "
        << "--network=none "
        << "--memory=" << limits.memoryLimitMb << "m
     "
        << "--cpus=" << limits.cpuLimit << " "
        << "--pids-limit=" << limits.pidsLimit << "
    "
        << volumeArg
        << imageName_ << " "
        << "/bin/bash -lc \"cd /workspace && "
        << "timeout " << timeLimitSeconds << "s "
        << "/usr/bin/time -v ./main < " <<
    inputFileName
        << " > " << outputFileName
        << " 2> " << runtimeLogName << "\"";

    int exitCode = std::system(cmd.str().c_str());
    result.exitCode = exitCode;

    // timeout devuelve 124 = TLE
    if (exitCode == 124) {
        result.timedOut = true;
    }

    return result;
}
```

**Comparación tolerante de salidas:**

Listing 12. Comparador de salidas

```
class OutputComparer {
public:
    static bool areEqual(
        const std::filesystem::path& outputFile,
        const std::filesystem::path& expectedFile)
    {
        // Implementa comparacion tolerante:
        // - Ignora \r
        // - Remueve espacios al final de lineas
        // - Ignora lineas vacias al final
        // Evita falsos WA por diferencias triviales
    }
};
```

**Orquestación del servicio:**

Listing 13. Flujo de evaluación

```
EvaluationResult EvaluationService::evaluate(
    const SubmissionRequest& request)
{
    EvaluationResult result;
    result.submissionId = request.submissionId;

    // 1. Crear directorio de submission
    auto submissionDir = SubmissionFilesystem::
        createSubmissionDir(baseDir_,
            request.submissionId);
```

```
    // 2. Escribir archivo fuente
    SubmissionFilesystem::writeSourceFile(
        submissionDir, "main.cpp",
        request.sourceCode);

    // 3. Escribir archivos de test
    SubmissionFilesystem::writeTestFiles(
        submissionDir, request.testCases);

    // 4. Compilar
    DockerRunner runner(dockerImage_);
    auto comp = runner.compile(submissionDir,
        "main.cpp");

    if (comp.exitCode != 0) {
        result.overallStatus =
            OverallStatus::CompilationError;
        return result;
    }

    // 5. Ejecutar cada test case
    for (const auto& tc : request.testCases) {
        TestResult tr = runAndCompare(runner,
            submissionDir, tc);
        result.tests.push_back(tr);
    }

    // 6. Determinar estado global
    result.overallStatus = determineOverallStatus(
        result.tests);

    return result;
}
```

**Extracción de métricas de memoria:**

Listing 14. Extracción de memoria usada

```
int extractMaxMemoryKb(const std::string& logText) {
    std::istringstream iss(logText);
    std::string line;
    const std::string key =
        "Maximum resident set size (kbytes):";

    while (std::getline(iss, line)) {
        auto pos = line.find(key);
        if (pos != std::string::npos) {
            std::string valueStr =
                line.substr(pos + key.size());
            return std::stoi(valueStr);
        }
    }
    return 0;
}
```

**Imagen Docker utilizada:**

Listing 15. Dockerfile para compilación

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y \
    g++ \
    time \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /workspace
```

**Alternativas consideradas:**

Se evaluaron alternativas como ejecutar código directamente con `fork()` y límites con `setrlimit()`, pero Docker fue preferido por su aislamiento más robusto (filesystem, red, procesos) y facilidad de configuración de límites.

**Limitaciones conocidas:**

- Solo soporta C++ como lenguaje.
- Requiere Docker instalado en el sistema host.
- La medición de memoria depende de `/usr/bin/time`.

*II-C3. Interacción en la UI:*

1. El usuario escribe su solución en el editor de código.
2. Presiona el botón “Run”.
3. La UI envía POST a `/submissions` del Gestor.
4. El Gestor reenvía a `/evaluate` del Motor (puerto 8090).
5. El Motor compila y ejecuta en Docker.
6. Los resultados se muestran test por test en la consola.
7. Se indica tiempo, memoria y estado (Accepted/WrongAnswer/etc).

### *II-D. Analizador de Soluciones*

*II-D1. Descripción del Requerimiento: “Es un REST API aparte que recibe el código ingresado por el usuario y los resultados del motor de evaluación. Se comunica con el LLM para proveer feedback, explicación de error, sugerencias, entre otros. Se debe ajustar la petición para no proveer una solución completa al usuario. Provee ayudas, pero manteniendo el nivel de reto.”*

*II-D2. Implementación del Requerimiento:* El Analizador de Soluciones combina análisis estático del código con inteligencia artificial (Gemini API) para proporcionar retroalimentación educativa sin revelar soluciones completas.

**Estructuras de datos utilizadas:**

Listing 16. Resultado del análisis

```
struct AnalysisResult {
    bool success;
    string complexity;        // O(n), O(n^2), etc.
    string algorithmType;     // Binary Search, etc.
    int nestedLoops;          // Profundidad de loops
    bool isRecursive;
    double averageRatio;      // Ratio de crecimiento
    vector<double> executionTimes;
    string explanation;
    vector<string> suggestions;
    string errorMessage;
    string consoleOutput;
};
```

**Análisis estático de complejidad:**

Listing 17. Conteo de loops anidados

```
int ComplexityAnalyzer::countNestedLoops() {
    int maxNivel = 0;
    int nivelActual = 0;
    bool dentroDeLoop = false;

    for (size_t i = 0; i < code.length(); i++) {
        // Detectar inicio de loop
        if (i + 3 < code.length()) {
            string sub = code.substr(i, 3);
            if (sub == "for" || sub == "whi") {
                dentroDeLoop = true;
            }
        }

        if (code[i] == '{' && dentroDeLoop) {
            nivelActual++;
            maxNivel = max(maxNivel, nivelActual);
        } else if (code[i] == '}') {
            if (nivelActual > 0) nivelActual--;
        }
```

```
    }
    return maxNivel;
}
```

### Identificación de patrones algorítmicos:

Listing 18. Identificación de tipo de algoritmo

```
string ComplexityAnalyzer::identifyAlgorithmType() {
    string codeLower = code;
    transform(codeLower.begin(), codeLower.end(),
        codeLower.begin(), ::tolower);

    map<string, vector<string>> patterns = {
        {"Linear Search",
            {"for", "arr[i]", "==", "return"}},
        {"Binary Search",
            {"while", "low", "high", "mid"}},
        {"Bubble Sort",
            {"for", "for", "swap"}},
        {"Merge Sort",
            {"merge", "mid", "recursiv"}},
        {"Quick Sort",
            {"pivot", "partition"}},
        {"Dynamic Programming",
            {"dp", "memo", "table"}},
    };

    for (const auto& [algorithm, keywords] :
    patterns) {
        int score = 0;
        for (const auto& keyword : keywords) {
            if (codeLower.find(keyword) !=
                string::npos) {
                score++;
            }
        }
        if (score >= 1) return algorithm;
    }

    return "Custom Algorithm";
}
```

### Determinación de complejidad:

Listing 19. Determinación de complejidad Big-O

```
string ComplexityAnalyzer::determineComplexity(
    const vector<double>& times)
{
    if (times.size() < 2) {
        // Analisis estatico
        if (detectRecursion()) return "O(2^n)";

        int loops = countNestedLoops();
        switch (loops) {
            case 0: return "O(1)";
            case 1: return detectLogarithmic() ?
                "O(log n)" : "O(n)";
            case 2: return "O(n^2)";
            case 3: return "O(n^3)";
            default: return "O(n^" +
                to_string(loops) + ")";
        }
    }

    // Analisis empirico con tiempos
    double avgRatio = 0;
    for (size_t i = 1; i < times.size(); i++) {
        if (times[i-1] > 0) {
            avgRatio += times[i] / times[i-1];
        }
    }
    avgRatio /= (times.size() - 1);

    if (avgRatio < 1.5) return "O(log n)";
    else if (avgRatio <= 2.3) return "O(n)";
    else if (avgRatio <= 2.5) return "O(n log n)";
    else if (avgRatio <= 4.5) return "O(n^2)";
    else return "O(2^n)";
}
```

### Integración con Gemini API:

Listing 20. Cliente de Gemini para sugerencias

```
vector<string> GeminiClient::generateSuggestions(
    const string& code,
    const string& complexity,
    const string& algorithmType,
    const string& consoleOutput)
{
    vector<string> suggestions;

    // Construir prompt educativo
    stringstream prompt;
    prompt << "Eres un asistente experto en "
           << "optimizacion de codigo C++.\n\n";
    prompt << "Codigo a analizar:\n```cpp\n"
           << code << "\n```\n\n";
    prompt << "Analisis:\n";
    prompt << "- Complejidad: " << complexity << "\n
    ";
    prompt << "- Tipo: " << algorithmType << "\n\n";
    prompt << "TAREA:\n";
    prompt << "Proporciona 3 sugerencias especificas
      "
           << "para mejorar este codigo.\n";
    prompt << "Enfocate en:\n";
    prompt << "1. Eficiencia del algoritmo\n";
    prompt << "2. Estructuras de datos\n";
    prompt << "3. Tecnicas de optimizacion\n";
    prompt << "NO des soluciones completas.\n";

    // Configurar request
    json payload = {
        {"contents", json::array({
            {{"parts", json::array({
                {{"text", prompt.str()}}
            })}}
        })},
        {"generationConfig", {
            {"temperature", 0.7},
            {"topK", 40},
            {"maxOutputTokens", 1024}
        }}
    };

    // Llamar API via CURL
    string url = apiUrl + "?key=" + apiKey;
    // ... realizar request y parsear respuesta

    return suggestions;
}
```

### Servidor REST del Analizador:

Listing 21. Endpoint principal del Analizador

```
CROW_ROUTE(app, "/api/analyze")
    .methods(crow::HTTPMethod::Post)
([geminiKey](const crow::request& req) {
    crow::json::wvalue response;

    auto body = crow::json::load(req.body);
    string code = body["code"].s();

    // Crear analizador con API key
    ComplexityAnalyzer analyzer(code, geminiKey);
    AnalysisResult result = analyzer.analyze();

    // Construir respuesta
```

```
    response["success"] = result.success;
    response["complexity"] = result.complexity;
    response["algorithmType"] = result.algorithmType
    ;
    response["details"]["nestedLoops"] =
        result.nestedLoops;
    response["explanation"] = result.explanation;

    for (size_t i = 0; i <
         result.suggestions.size(); i++) {
        response["suggestions"][i] =
            result.suggestions[i];
    }

    crow::response res(response);
    res.add_header("Access-Control-Allow-Origin", "*
    ");
    return res;
});
```

**Carga de variables de entorno:**

Listing 22. Cargador de archivo .env

```
EnvLoader::EnvLoader(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "[ADVERTENCIA] " << filename
             << " no encontrado" << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        if (line.empty() || line[0] == '#') continue
    ;

        size_t pos = line.find('=');
        if (pos == string::npos) continue;

        string key = line.substr(0, pos);
        string value = line.substr(pos + 1);

        // Trim y remover comillas
        // ...

        envVars[key] = value;
    }
}
```

**Alternativas consideradas:**

Se evaluaron otras APIs de LLM como OpenAI GPT-4 y Claude, pero Gemini fue seleccionado por su capa gratuita generosa y su API simple de integrar con C++/libcurl.

**Limitaciones conocidas:**

- El análisis estático es heurístico y puede no detectar todos los patrones.
- La calidad de las sugerencias depende del modelo Gemini.
- Requiere conexión a Internet para las sugerencias AI.

*II-D3. Interacción en la UI:*

1. El usuario escribe su solución y presiona "Submit".
2. La UI envía POST a `/api/analyze` del Analizador.
3. El Analizador realiza análisis estático del código.
4. Se consulta Gemini API para generar sugerencias.
5. La consola muestra:
   - Complejidad detectada (ej: $O(n^2)$)
   - Tipo de algoritmo identificado
   - Número de loops anidados
   - Explicación de la complejidad
   - Sugerencias de mejora (sin dar soluciones)

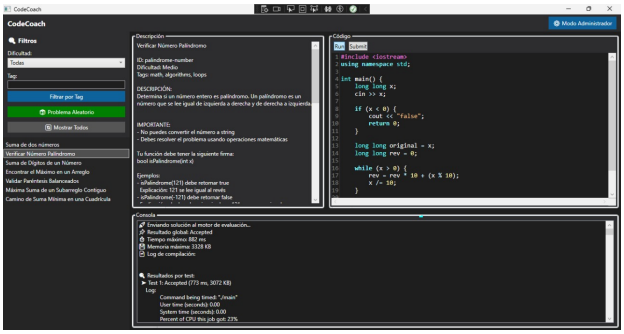

Figura 3. Feedback del analizador mostrando detalles, explicación y sugerencias de Gemini

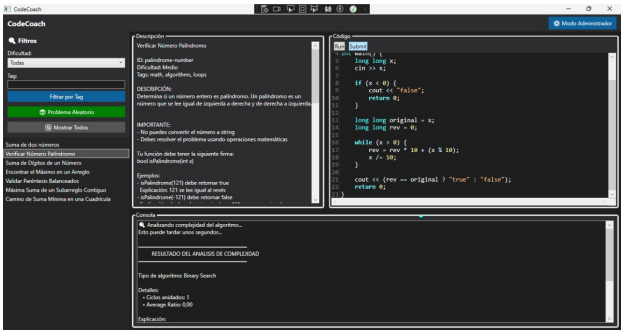

Figura 4. Feedback del analizador mostrando complejidad

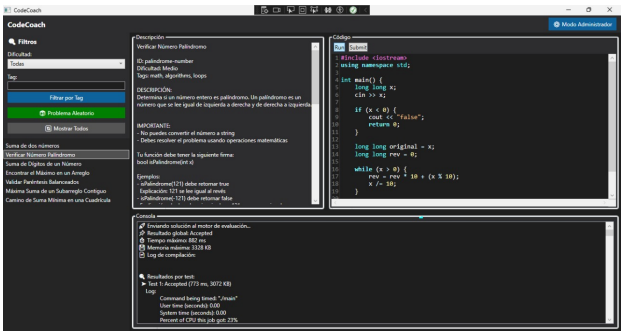

Figura 5. Estado de aceptación de analizador

## III. Arquitectura general de CodeCoach

El siguiente diagrama muestra las clases principales del sistema y sus relaciones:

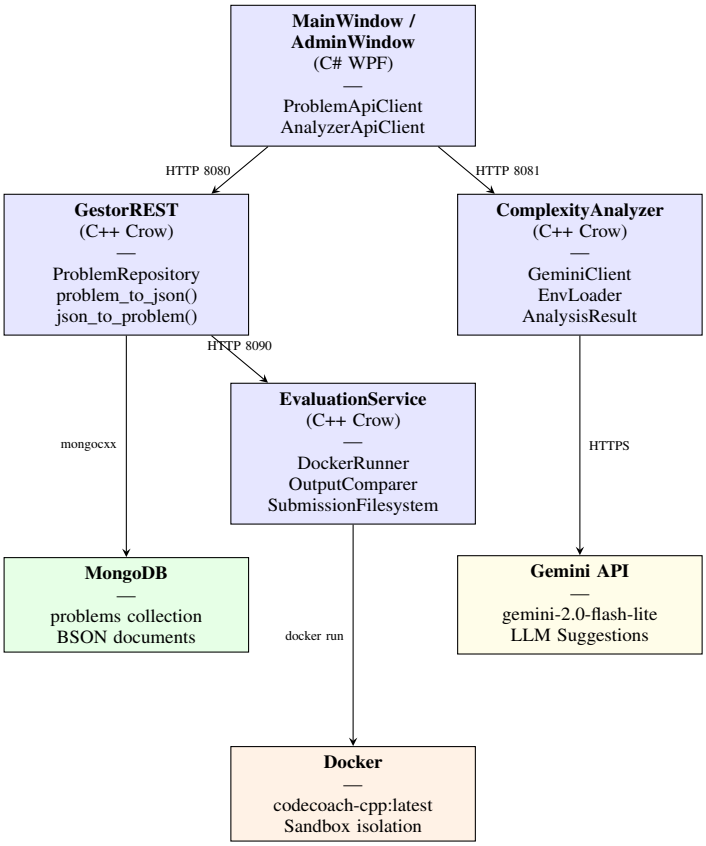


Figura 6. Diagrama reducido de la arquitectura de CodeCoach

### *III-A. Patrones de Diseño Aplicados*

**Repository Pattern:** Implementado en `ProblemRepository` para encapsular el acceso a MongoDB, separando la lógica de negocio de la capa de persistencia.

**Strategy Pattern:** El `ComplexityAnalyzer` usa diferentes estrategias para determinar la complejidad (análisis estático vs. empírico).

**Facade Pattern:** `EvaluationService` actúa como fachada que orquesta `DockerRunner`, `OutputComparer` y `SubmissionFilesystem`.

**Adapter Pattern:** Las clases `ProblemApiClient` y `AnalyzerApiClient` en C# adaptan las respuestas HTTP a objetos fuertemente tipados.

**Proxy Pattern:** El Gestor actúa como proxy entre la UI y el Motor de Evaluación para el endpoint `/submissions`.

## IV. Enlace al Repositorio

El código fuente completo del proyecto está disponible en el siguiente repositorio de GitHub:

https://github.com/habycr/Proyecto_2_Datos_2_IIS_2025.git

## V. Prompts Utilizados

A continuación se documentan los prompts utilizados para la asistencia de agentes de inteligencia artificial durante el desarrollo del proyecto:

### *V-A. Prompts para Generación de Código*

1. **Arquitectura del Motor de Evaluación:** "Necesito diseñar un motor de evaluación de código en C++ que ejecute código de usuarios en un sandbox seguro usando Docker. El motor debe compilar código C++, ejecutar tests con límites de tiempo y memoria, y comparar salidas. Dame la arquitectura de clases y los métodos principales que debería tener para poder implementarlo correctamente."
2. **Integración con MongoDB:** "Cómo implemento un Repository Pattern en C++ usando mongocxx para gestionar documentos de problemas de programación con campos: problem_id, title, description, difficulty, tags (array), test_cases (array de objetos)."
3. **REST API con Crow:** "Necesito crear endpoints REST en C++ usando Crow framework para un CRUD completo de problemas. Incluye validación de JSON, manejo de errores y respuestas apropiadas. ¿Cómo lo implemento y cómo se usa CROW para este fin?"
4. **Cliente HTTP en C#:** "Debo implementar un cliente HTTP en C# para consumir una API REST de problemas de programación. Debe soportar operaciones CRUD con manejo de errores y deserialización JSON. ¿Cómo se hace?"
5. **Análisis de Complejidad:** "Necesito diseñar un analizador de complejidad algorítmica en C++ que detecte loops anidados, recursión, y patrones comunes de algoritmos. Debe estimar la complejidad Big-O basándose en análisis estático del código. ¿Qué herramientas debo utilizar para poder realizar esta tarea?"

### *V-B. Prompts para Integración de APIs*

6. **Integración Gemini API:** "Cómo llamo a la API de Gemini desde C++ usando libcurl para obtener sugerencias de optimización de código sin revelar soluciones completas. Dame una guía completa"
7. **Comunicación Docker desde C++:** "Cómo ejecuto comandos Docker desde C++ para compilar y ejecutar código con límites de recursos (memoria, CPU, tiempo ) y capturar stdout/stderr. Dame una guía paso a paso"

### *V-C. Prompts para Debugging*

8. **Problemas de CORS:** "Mi API C++ (Crow) rechaza peticiones desde la UI WPF. Cómo configuro los headers CORS correctamente para permitir peticiones cross-origin."
9. **Parsing JSON BSON:** "Tengo varios errores al convertir documentos BSON de MongoDB a structs C++. El campo test_cases es un array de objetos y se logran iterar correctamente."
10. **Timeout Docker:** "El comando docker run no respeta el timeout que le paso. El código del usuario puede ejecutarse indefinidamente. Cómo implemento un timeout real."

### *V-D. Prompts para Documentación*

11. **Diagramas TikZ:** “Crea un diagrama TikZ que muestre la arquitectura de microservicios de CodeCoach con UI, Gestor, Motor, Analizador, MongoDB, Docker y Gemini API.”

## REFERENCIAS


[1] Crow Framework, “A C++ micro web framework inspired by Python Flask,” 2024. [En línea]. Disponible: https://crowcpp.org/

[2] MongoDB, “MongoDB C++ Driver Documentation,” 2024. [En línea]. Disponible: https://www.mongodb.com/docs/drivers/cxx/

[3] Docker Inc., “Docker Documentation - Run containers,” 2024. [En línea]. Disponible: https://docs.docker.com/engine/reference/run/

[4] Google, “Gemini API Documentation,” 2024. [En línea]. Disponible: https://ai.google.dev/gemini-api/docs

[5] Microsoft, “Windows Presentation Foundation Documentation,” 2024. [En línea]. Disponible: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/

[6] AvalonEdit, “The WPF-based text editor component,” 2024. [En línea]. Disponible: https://github.com/icsharpcode/AvalonEdit

[7] N. Lohmann, “JSON for Modern C++,” 2024. [En línea]. Disponible: https://github.com/nlohmann/json

[8] LeetCode, “LeetCode - The World’s Leading Online Programming Learning Platform,” 2024. [En línea]. Disponible: https://leetcode.com/

[9] Big-O Cheat Sheet, “Know Thy Complexities!,” 2024. [En línea]. Disponible: https://www.bigocheatsheet.com/

[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley, 1994.

# ANEXOS

## *Anexo A: Diagrama UML Completo del Sistema*

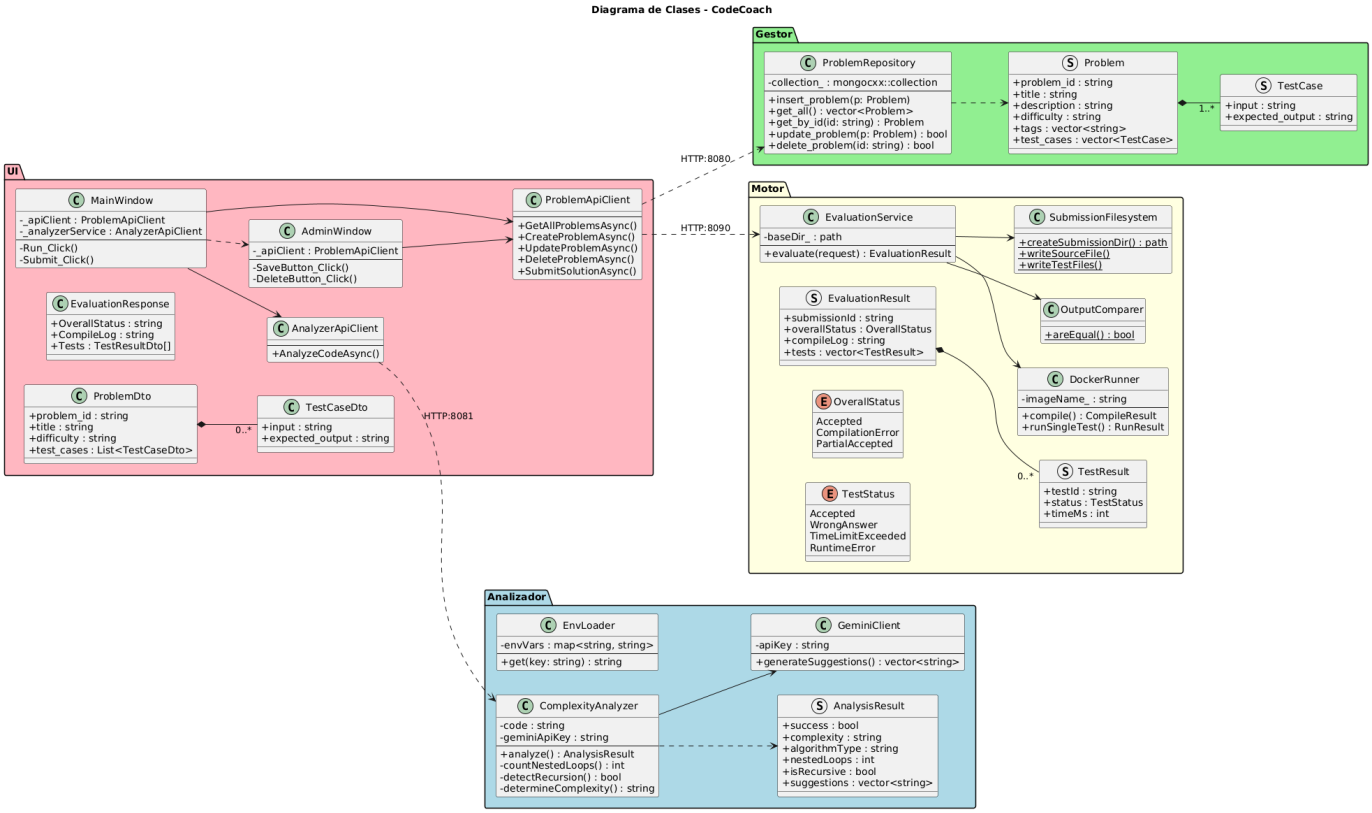


Figura 7. Diagrama UML completo de la arquitectura de CodeCoach mostrando todos los componentes del sistema: Interfaz Gráfica (C# WPF), Gestor de Problemas, Motor de Evaluación, Analizador de Soluciones, y sus interacciones con MongoDB, Docker y Gemini API