Technical Report – InsuraPro Solutions

Fortunato Cogliandro

1. Project Overview

This project implements a simple Customer Relationship Management (CRM) system for an insurance company. The goal was to build a console-based application in C++ that allows users to manage customer information and their related interactions such as appointments or contracts.

The program is developed using the principles of Object-Oriented Programming (OOP), focusing on class design, encapsulation, and modularity. All data is stored in memory using C++ Standard Template Library (STL) containers and persisted to CSV files for future use.

2. System Design

The CRM system is structured around three main classes:

- **Customer:** represents an individual client with attributes such as id, name, surname, email, and phone. It includes getter and setter methods to access and modify data safely, as well as a simple method to print customer details.
- **Interaction:** represents events or actions related to a customer, such as meetings or contracts. It contains attributes for id, customerId, type, date, and notes.
- **CRM:** the main controller class that manages both Customer and Interaction objects. It handles adding, listing, searching, modifying, and deleting customers, as well as managing their interactions.

Each class is implemented in separate .h and .cpp files to ensure modularity and readability.

3. Functionalities Implemented

The application provides the following features:

- **Add a Customer:**
  Allows inserting a new customer into the system. Before adding, the program checks for duplicate emails to avoid repeated records. Basic validation is applied to the email, phone number, and input fields.
- **List Customers:**
  Displays all registered customers in a clear format with their basic details.

- **Search Customer by Name or Surname:**
  Supports case insensitive search. It prints all customers whose name or surname matches the search query.

- **Modify a Customer:**
  Enables updating customer data (name, surname, email, or phone). The user can skip a field to keep the current value.
- **Delete a Customer:**
  Removes a customer from the system and automatically deletes all interactions linked to that customer.
  Note: customer IDs are auto incremented and never reused, even after deletion. This ensures that interactions linked to deleted customers remain consistent and avoids potential ID conflicts when loading data from CSV files.
- **Add an Interaction:**
  Associates an interaction with an existing customer. The system checks that the customer ID exists before adding the interaction and validates the date format (DD/MM/YYYY).
- **List Interactions:**
  Lists all interactions associated with a specific customer.
- **Data Persistence:**
  Customer and interaction data are stored in customers.csv and interactions.csv. The system loads data automatically when it starts and saves updates on exit.

4. Data Handling and Validation

The data is stored using std::vector containers for simplicity and efficiency. Input validation was implemented for:

- Email format: must contain '@' and '.'
- Date format: must follow DD/MM/YYYY and be within valid day/month ranges
- Phone number: must contain digits only

These validations make the program more realistic and robust, even if it remains lightweight.

5. User Interface

The interface is completely console based, using a clear text menu. Users can navigate through options like adding, listing, modifying, and searching customers or interactions. Input errors are handled and invalid entries trigger simple warning messages. The interface was intentionally kept minimal and easy to use.

References & Resources

- C++ Reference – cppreference.com
- Object Oriented Programming in C++
- cplusplus.com – std::vector Reference
- C++ File Handling
- Header files (C++)
- C++ Classes and Objects
- Various YouTube videos
- ChatGPT – used for reviewing grammar and readability of the documentation