**1. Analisi**

**1.1 Descrizione del Sistema**

Il sistema riproduce le funzionalità di una piattaforma di gestione dati immobiliari. Il programma dovrà permettere di leggere da file e visualizzare i dati dei clienti, professionisti ed immobili, consentire quindi l’aggiunta di nuovi clienti. Dovrà inoltre consentire la ricerca tra gli immobili presenti in memoria in basi a criteri quali prezzo o località ed inoltre la visualizzazione dell’agenzia entro un certo intervallo di tempo.

Ogni entità (clienti, professionisti e immobili) è stata definita con i loro attributi caratteristici.

Sono state implementate diverse funzioni aggiuntive quali ordinamento alfabetico dei file e ricerca di duplicati con possibilità di risoluzione del conflitto direttamente all’interno del programma.

**1.2 Requisiti Funzionali**

|  |  |  |
| --- | --- | --- |
| Codice | Nome | Descrizione |
| R01 | Visualizzazione Menu | Il programma deve mostrare all’utente un menu iniziale dove vengono visualizzate le opzioni disponibili. |
| R02 | Caricamento Dati da File | Il programma deve caricare i dati da file per ogni entità. |
| R03 | Ordinamento alfabetico dei File | Il programma deve garantire il corretto ordinamento dei file. |
| R04 | Ricerca Duplicati | Il programma deve cercare eventuali conflitti tra i dati presenti e risolverli. |
| R05 | Ricerca Immobili | Il programma deve visualizzare gli immobili che rispettano determinati criteri decisi dall’utente. |
| R06 | Riscrittura File | Il programma deve riscrivere i file con le informazioni aggiornate. |
| R07 | Aggiunta clienti | Il programma deve poter aggiungere nuovi clienti con dati inseriti dall’utente. |
| R08 | Controllo clienti scaduti | Il programma deve controllare la data e permettere la cancellazione dell’utente se scaduto. |
| R09 | Visualizzazione Dati | Il programma deve mostrare le informazioni sugli immobili, clienti e professionisti. |

**1.3 Strumenti di Sviluppo**

Il sistema è stato realizzato utilizzando un PC Desktop Intel i7-4771 a 3.5 GHZ, con 16GB di Ram. L’ambiente di sviluppo adottato è stato Eclipse, corredato dal plugin Eclox per documentazione con Doxygen. Per eseguire il sistema è necessario avere un computer con sistema operativo Windows.

**2. Progettazione**

**2.1 Progettazione dei tipi di dato, delle strutture dati**

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Tipologia | Descrizione | Tipi / Campi / Valori |
| Client | struct | Tipo di dato definito per descrivere le caratteristiche del cliente | ID:  char[50]  Nome: char[50]  Cognome: char[50]  Tipo cliente:  client\_type  Nome compagnia:  char[50]  Budget:  int  Data registrazione:  time\_t  Tipo immobile ricercato:  building\_type  Da eliminare:  bool |
| Client\_type | enum | Tipo di Dato che descrivi i tipi di cliente | family / single / company / government |
| Building\_type | enum | Tipo di Dato che descrivi i tipi di immobile | Flat / duplex / house / farmhouse / attic |
| professional | struct | Tipo di Dato che descrivi i tipi di professionisti | ID:  char[50]  Nome: char[50]  Cognome: char[50]  Area di competenza:  char[50]  Numero di telefono:  char[50]  E-mail:  char[50]  Data registrazione:  time\_t  Immobili venduti:  int |
| potential | struct | Tipo di Dato che descrive il potenziale dei professionisti | ID:  char[50]  Contenuto:  char[50] |
| building | struct | Tipo di Dato che descrivi i tipi di immobile | ID:  char[50]  Via:  char[50]  Civico:  short int  Città:  char[50]  Provincia:  char[50]  Data registrazione:  time\_t  Prezzo:  int  Proprietario:  char[50]  N. telefono proprietario:  char[50]  Tipo di immobile:  building\_type |

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Tipologia | Descrizione | Tipi / Campi / Valori |
| STRING\_SIZE | costante | Costante utilizzata per memorizzare la dimensione delle stringhe | 50 |
| MIN\_USER\_BUDGET | costante | Costante utilizzata per memorizzare il numero minimo di budget | 100 |
| MAX\_USER\_BUDGET | costante | Costante utilizzata per memorizzare il numero massimo di budget | 1000000000 |
| buildings.csv | File | File utilizzato per salvare le info sugli immobili |  |
| DAY\_IN\_SECONDS | costante | Costante utilizzata per memorizzare il numero di secondi in un giorno | 86400 |
| CLIENT\_EXPIRE\_DAYS | costante | Costante utilizzata per memorizzare il numero di giorni prima della scadenza | 30 |
| Clients.csv | File | File utilizzato per salvare le info sui clienti |  |
| Potential.csv | File | File utilizzato per salvare le info sui potenziali |  |
| Professionals.csv | File | File utilizzato per salvare le info sui professionisti |  |

**2.2 Progettazione delle librerie / funzioni**

<Indicare quali sono le librerie progettate. Per ciascun file .h indicare le procedure e le funzioni incluse nell’header file. Per ogni funzione indicare scopo, tipi di ingresso e di uscita. In caso di uso di Doxygen, limitarsi a elencare le librerie, i metodi di ciascuna libreria e il criterio di progettazione della libreria, e rimandare alla documentazione Doxygen le altre informazioni>

|  |  |
| --- | --- |
| Libreria | Metodi |
| client.h | void newLine();  void clearScr();  bool isNumber(char \*str);  bool isChar(char \*str);  void printFormattedDate(time\_t epochTime);  void printSectionName(char \*headerString);  void convertToUpperCase(char \*s);  int readString(char \*value, bool onlyAlpha);  int readInteger();  bool strCompare(char \*from, char \*to);  bool askConfirm();  void resetColor();  void setRedColor();  void setYellowColor();  void setGreenColor();  void setCyanColor();  void setMagentaColor();  void pause();  void notFoundError(); |
| sort.h | void sortFilePro(professional \*pr, int rows);  void sortFileCli(client \*cl, int rows);  void sortFileBui(building \*bl, int rows); |
| menus.h | void mainMenu();  int clientsMenu(int jump);  int professMenu();  int buildingsMenu(); |
| file\_utils.h | bool checkFile(FILE \*filePtr, bool rowsCheck);  int countRows(FILE \*filePtr);  void copyFile(FILE \*fp\_from, FILE \*fp\_to, int choice);  time\_t parseDateInFile(char token[STRING\_SIZE]); |
| agency.h | int resultAg(building \*bl, int n\_bui); |
| files\_pr.h | int loadProsFile(professional \*pr);  int getProfessionalsNumber();  int rewriteProfessionalsToFile(professional \*pr, int rows);  void readProsFile(FILE \*filePtr, professional \*pr);  void readPotFile(FILE \*fp\_pot, potential \*pr, char id[], int rows);  void findPot(char id[], potential \*pr, int rows);  int loadPotFile(char id[]);  int checkDuplicatePro(professional \*pr, int rows); |
| misc\_pr.h | void initPotentialsArray(potential \*po, int size);  void initProsArray(professional \*pro, int size); |
| show\_pr.h | void showProData(professional \*pr);  int showAllPros(professional \*pros, int num\_pros); |
| files\_cl.h | int loadClientFile(client \*cl);  void parseClientFile(FILE \*filePtr, client \*cl);  int rewriteClientsToFile(client \*cl, int rows);  int appendClientToFile(client \*cl);  int checkDuplicateClients(client \*cl, int rows);  int getClientsNumber(); |
| misc\_cl.h | int addClient();  void saveLocalDate(client \*cl);  bool checkIfUserExpired(time\_t epochTime, char id[]);  void initClientsArray(client \*cl, int size); |
| req\_cl.h | void reqID(client \*cl);  void reqCF(client \*cl);  void reqPIVA(client \*cl);  void reqName(client \*cl);  void reqSurname(client \*cl);  void reqType(client \*cl);  void reqCompanyName(client \*cl);  void reqBudget(client \*cl);  void reqPropertyType(client \*cl); |
| show\_cl.h | void showClientType();  void showClientData(client \*cl);  int showAllClients(client \*cl, int num\_clients); |
| file\_bl.h | int loadBuildingsFile(building \*bl);  void readBuildingsFile(FILE \*filePtr, building \*cl);  void searchBuilding(building \*bl, int n\_bui);  int getBuildingsNumber();  int rewriteBuildingsToFile(building \*bl, int rows);  int checkDuplicateBuildings(building \*bl, int rows); |
| misc\_bl.h | void initBuildingsArray(building \*bl, int size); |
| show\_bl.h | void showBuildingType();  void showBuildingData(building \*bl);  int showAllBuildings(building \*bl, int num\_buildings); |
| consts.h |  |
| datatypes.h |  |

**3. Codifica**

<Allegare la Documentazione Prodotta con Doxygen>

**4. Testing**

**4.1 Definizione del Piano di Test**

<Per ciascun caso d’uso definito in sezione 1.2 definire i casi di test e validarne l’esito. Quando possibile, progettare i casi di test come asserzioni CUnit>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Codice Requisito | Codice Test | Nome | Descrizione Test | Eventuale Input | Risultato Atteso | Risultato Ottenuto |
| R01 | 1.1 | Menu Iniziale | Scelta n.1 | 1 | Caricamento del menu X | TRUE |
| R02 | 1.2 | Menu iniziale | Scelta errata | 100 | Messaggio di Errore | FALSE |
| R02 | 2.1 | Caricamento Dati da File | File non esistente | xxx | Visualizzazione Messaggio di Errore e Creazione Nuovo File | FALSE |
| R02 | 2.2 | Caricamento Dati da File | File Esistente | xxx | Caricamento dei File effettuato corretamente | TRUE |
| R03 | 3.1 | Ordinamento File | Struttura in memoria | xxx | Ordinamento effettuato correttamente | TRUE |
| R04 | 4.1 | Ricerca duplicati | Struttura in memoria | xxx | Trova eventuali duplicati | TRUE |
| R05 | 5.1 | Ricerca immobili | Località presente | barletta | Visualizza immobili | TRUE |
| R05 | 5.2 | Ricerca immobili | Località non presente | barlstkt | Non restituisce immobili | FALSE |
| R06 | 6.1 | Riscrittura File |  | xxx | Riscrive il file correttamente | TRUE |
| R07 | 7.1 | Aggiunta clienti | Aggiunta corretta | Dati cliente | Aggiunge correttamente il nuovo cliente | TRUE |
| R08 | 8.1 | Controllo clienti scaduti | Presenza di clienti scaduti | s | Segnala il cliente scaduto e lo elimina | TRUE |
| R09 | 9.1 | Visualizzazione dati |  | xxx | Visualizza i dati correttamente | TRUE |

**4.2 Esiti del Piano di Test ed eventuali commenti**

<Commentare gli esiti del piano di test, individuare eventuali criticità (test che l’attuale implementazione non riesce a superare) e pianificare eventuali azioni migliorative sul codice>