2.3 Разработка шаблона кода

Данный пункт посвящен трансформации объектно-ориентированной модели в модульную архитектуру, пригодную для последующей программной реализации. Центральным элементом этого этапа является диаграмма компонентов, которая служит для логического структурирования ранее определенных классов в независимые программные модули.

Для обеспечения четкого разделения ответственности мы используем трехслойный архитектурный подход Boundary-Control-Entity. Данный подход позволяет эффективно распределить классы по трем основным компонентам системы «Регистрация и квалификация Лида»:

1. Компонент Boundary включает классы, ответственные за презентацию данных и взаимодействие с пользователем.
2. Компонент Control содержит управляющий класс, который инкапсулирует бизнес-правила и координирует взаимодействие между интерфейсом и данными.
3. Компонент Entity представляет классы, напрямую соответствующие постоянным сущностям базы данных, обеспечивая хранение и управление информацией [9].

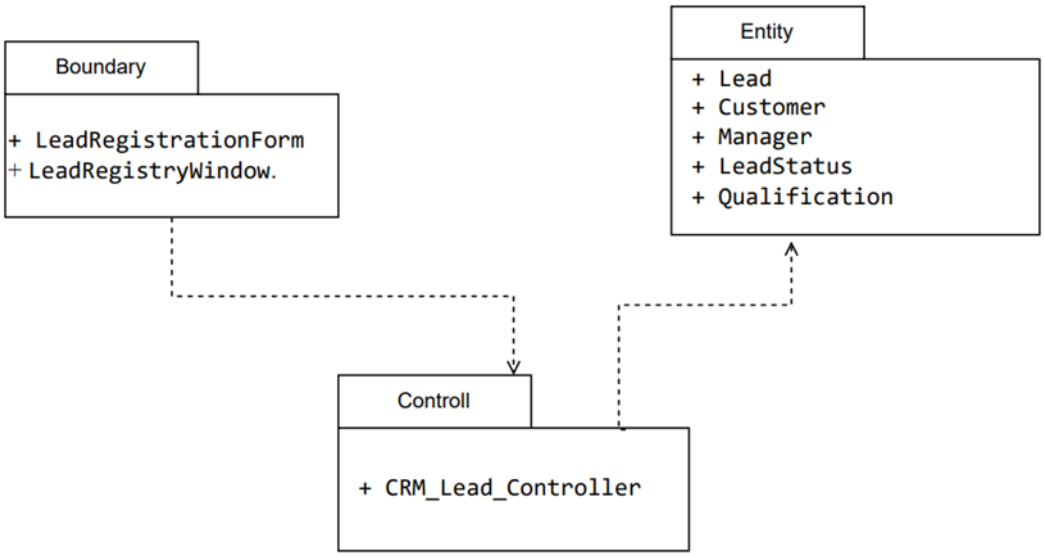

Рисунок 2.6 – Модульная структура компонентов информационной системы «Регистрация и квалификация Лида»

На основе данной модульной схемы автоматически формируются шаблонные файлы исходного кода. Далее представлен пример генерации заготовок на языке C++ для одного из наиболее важных элементов слоя данных (Entity) — класса «Lead».

```
#ifndef LEAD_ENTITY_DEFINITION_H
#define LEAD_ENTITY_DEFINITION_H

#include <string>
#include <ctime>
#include "Manager.h"
#include "LeadStatus.h"
#include "Qualification.h"

class Lead
{
public:
    Lead();
    virtual ~Lead();

    Manager *p_assignedManager;
    LeadStatus *p_currentStatus;
    Qualification *p_leadQualification;

private:
    int entity_id; // Первичный ключ
    std::string contact_fullName;
    std::string contact_emailAddress;
    std::string contact_phoneNumber;
    std::string financial_inn;
    std::string detail_description;
    std::time_t timestamp_creation;
    std::time_t timestamp_lastUpdate;

    void updateLeadState(int newStatusID);
    void linkToManager(int managerID);
};
```

Файл реализации Lead.cpp содержит определения методов класса, являясь отправной точкой для дальнейшего добавления конкретной бизнес-логики.

```
Lead::Lead(){
}
```

```
Lead::~Lead(){
}
void Lead::updateLeadState(int newStatusID){
}
void Lead::linkToManager(int managerID){
}
```