

## ГЛАВА 6. РАЗРАБОТКА ШАБЛОНА КОДА

В данной главе рассматривается процесс проектирования программных компонентов системы, а также формирование шаблонов кода, необходимых для последующей реализации. На основании ранее выполненного анализа предметной области, диаграмм вариантов использования, диаграмм классов и схемы данных выполняется переход от концептуального проектирования к проектированию конкретной архитектурной структуры будущего программного продукта.

Проектирование шаблона кода направлено на определение основных модулей системы, их взаимосвязей, а также на формирование начальных заготовок (скелетов) классов, являющихся фундаментом для разработки. Такой подход позволяет обеспечить:

- структурированность исходного кода;
- поддержку модульности и расширяемости;
- снижение связности между компонентами;
- соответствие архитектурным принципам проектирования.

На рисунке 6.1 представлена укрупнённая схема пакетов программной системы, отражающая разделение функциональности по слоям архитектуры.

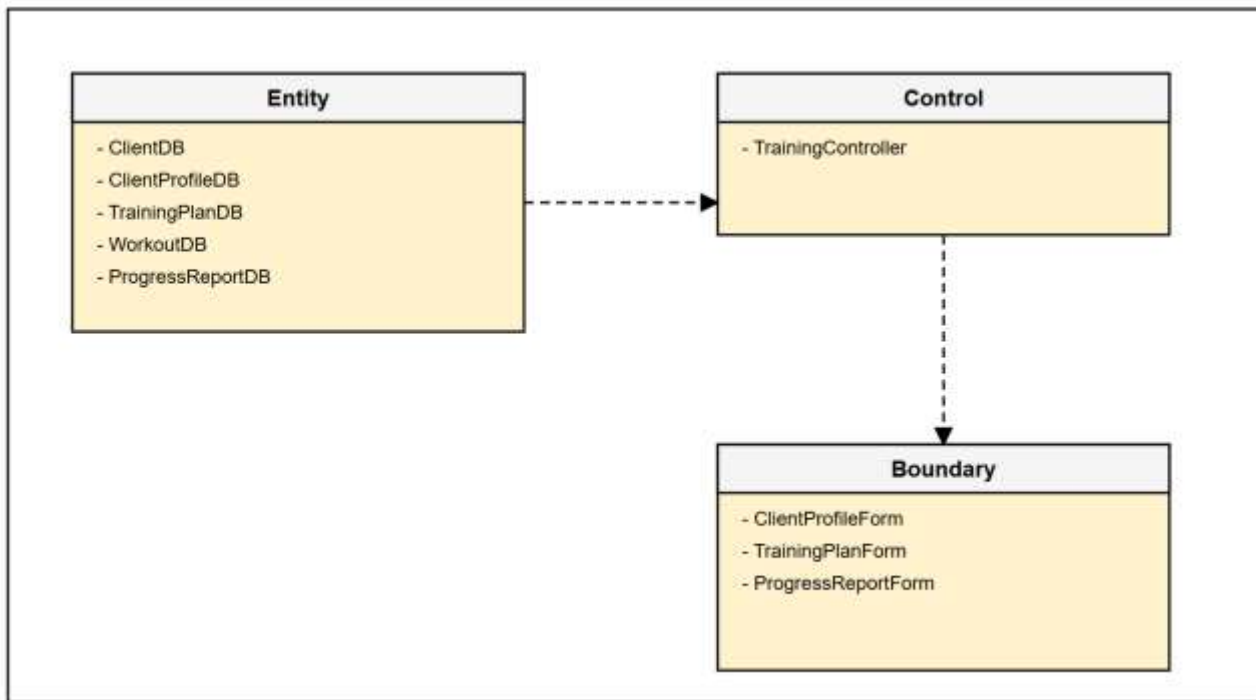


Рисунок 6.1 – Схема пакетов программной системы

Данная диаграмма выполнена в нотации UML (диаграмма пакетов) и демонстрирует взаимодействие между слоями Entity, Control, Boundary при разработке программного решения по учёту и проведению тренировок.

Пакет Entity содержит ключевые сущности и структуры данных, обеспечивающие хранение и доступ к информации в системе. В него включены следующие компоненты:

- ClientDB — хранение данных о клиентах;
- ClientProfileDB — сохранение профилей клиентов (цели, уровень подготовки, ограничения);
- TrainingPlanDB — данные индивидуальных планов тренировок;
- WorkoutDB — данные выполненных тренировок и отчётов о прогрессе.

Этот пакет представляет собой «нижний» уровень системы и не зависит от логики управления или интерфейса. На диаграмме видно, что пакет не имеет исходящих стрелок, что отражает правильный принцип однонаправленных зависимостей.

Пакет Control содержит прикладную логику системы. В него входит основной контроллер:

- TrainingController — координирует обработку профиля клиента, подбор тренировочной программы, расчёт прогресса и формирование отчёта.

Контроллер зависит от пакета Entity, что отражено пунктирной стрелкой: он использует хранимые данные, но не предоставляет им прямого доступа к своей внутренней логике. Это соответствует принципам многослойной архитектуры.

Пакет Boundary включает пользовательские формы, обеспечивающие визуальное взаимодействие пользователя с системой:

- ClientProfileForm — ввод и редактирование профиля клиента;
- TrainingPlanForm — подбор упражнений и формирование плана тренировок;
- ProgressReportForm — просмотр отчётов о прогрессе и учёт посещений.

Пакет Boundary зависит от пакета Control, что отражено направленной пунктирной стрелкой вниз: формы вызывают методы контроллера, но не обращаются напрямую к данным.

Рисунки 6.2, 6.3, 6.4 представляют собой детализированную декомпозицию системы на три ключевых пакета: Boundary, Entity и Control, выполненную согласно нотации UML. Такая структура отражает архитектурный подход,

направленный на разделение ответственности между различными уровнями системы.

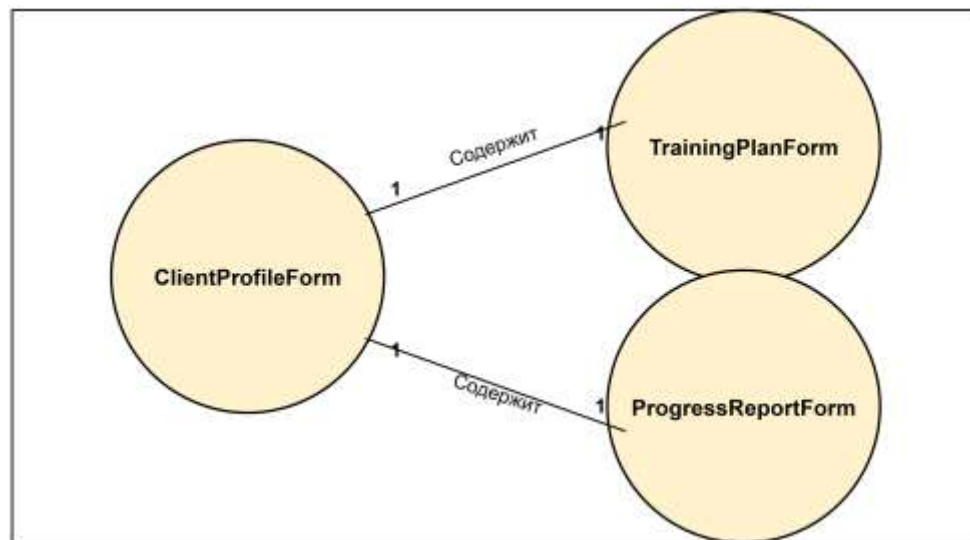


**Рисунок 6.2 – Пакет Boundary**

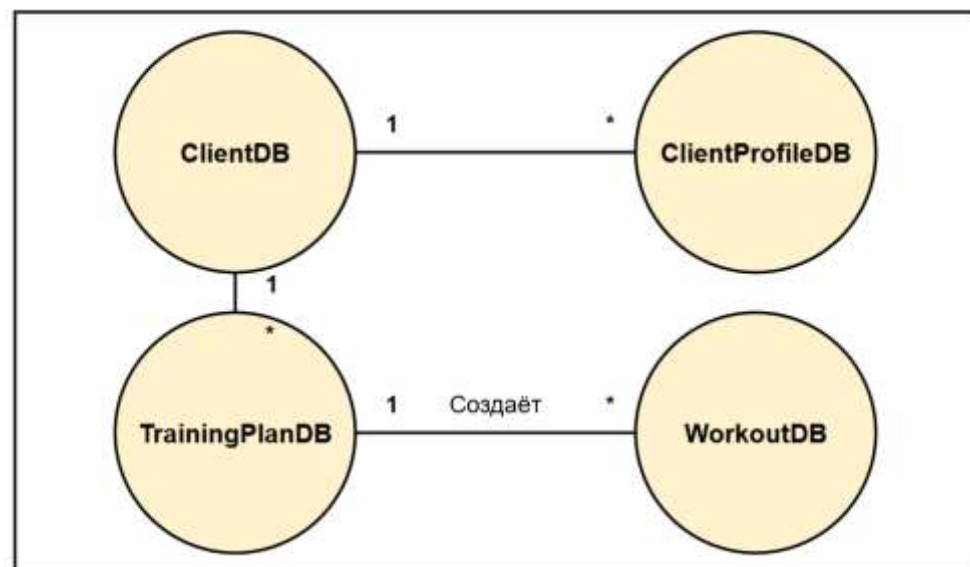


**Рисунок 6.3 – Пакет Entity**

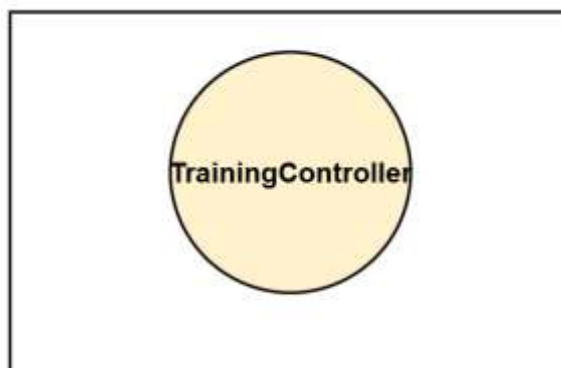


**Рисунок 6.4 – Пакет Control**

После формирования диаграмм была построена диаграмма компонентов, представленная на рисунке 6.5.

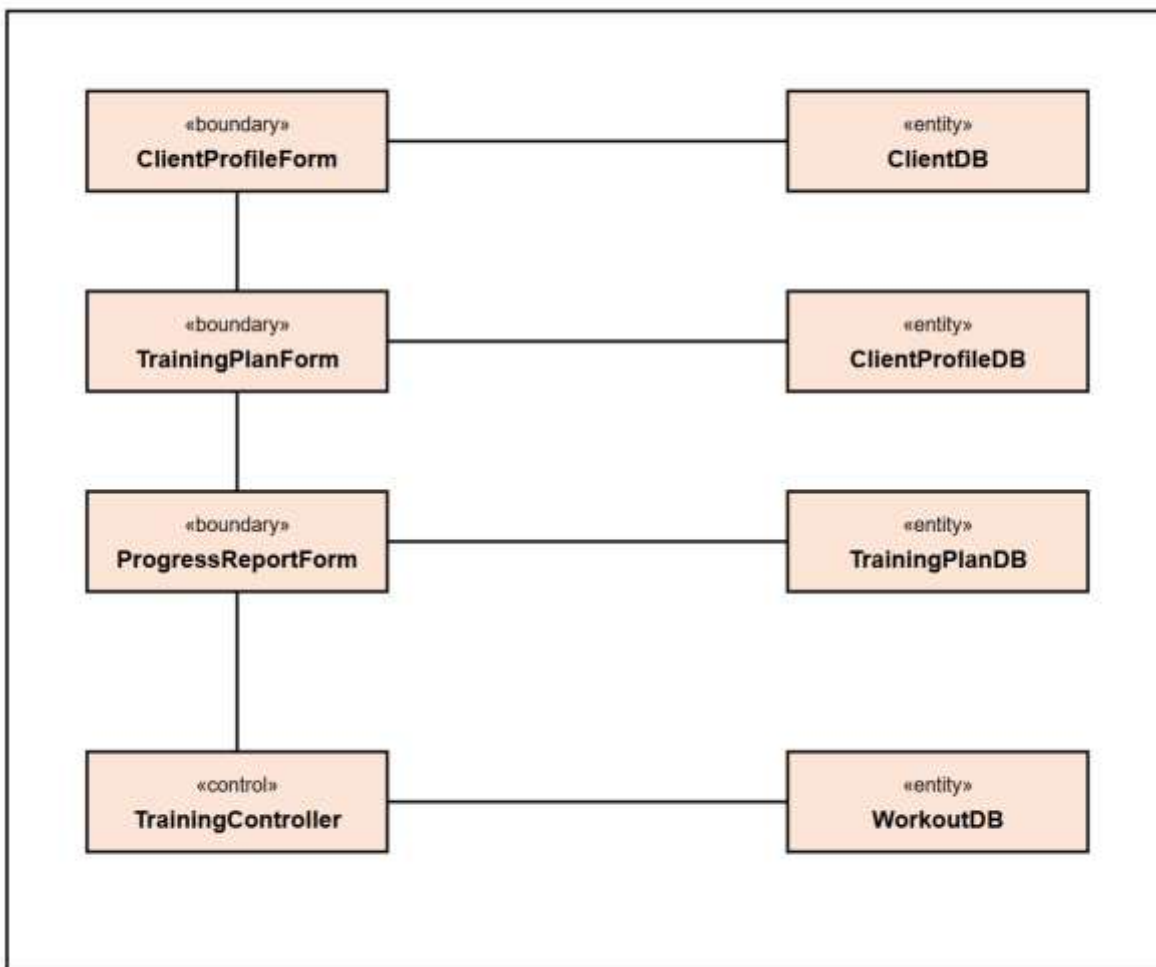


**Рисунок 6.5 – Диаграмма компонентов**

Диаграмма отражает архитектурную структуру разрабатываемой информационной системы и показывает взаимосвязи между boundary-компонентами, entity-компонентами и контроллером.

В состав системы входят три boundary-компонента (ClientProfileForm, TrainingPlanForm, ProgressReportForm), обеспечивающие взаимодействие пользователя с системой, ввод данных, просмотр планов и отчётов.

Все операции, требующие бизнес-логики, обрабатываются компонентом управления TrainingController, который реализует связь между пользовательскими формами и уровнями данных.

Хранение информации осуществляется в entity-компонентах (ClientDB, ClientProfileDB, TrainingPlanDB, WorkoutDB), которые инкапсулируют доступ к данным и обеспечивают структурную целостность системы.

Связи между компонентами минимизированы — на диаграмме задействован каждый элемент, но исключены избыточные зависимости. Такой подход соответствует требованиям модульности, расширяемости и низкой связанности.

Ниже приведён пример кода, сформированного для компонента TrainingPlanForm, который взаимодействует с контроллером и инициирует создание плана тренировок.

```
#include "TrainingController.h"
#include "ClientProfileDB.h"
#include "TrainingPlanDB.h"

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

    void set_controller(TrainingController *controller);
    void load_profile(int profileID);
    void submit_plan();
    void set_version(int v);

private:
```

```

    TrainingController *m_Controller;
    ClientProfileDB *m_ProfileDB;
    TrainingPlanDB *m_PlanDB;
    int version;
    char clientName[50];
};

```

Также представлен код и для остальных компонентов. Например, фрагмент структуры данных для ClientProfileDB:

```

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

    void save(ProfileData data);
    ProfileData get(int profileID);

private:
    int clientID;
    char goal[100];
    char level[50];
};

```

Контроллер, представленный на диаграмме как компонент управления, также представлен:

```

#include "TrainingPlanDB.h"
#include "ClientProfileDB.h"

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

    void generate_plan(int profileID);
    void approve(int planID);
};

```

```
void update(int planID);

private:
    TrainingPlanDB *m_PlanDB;
    ClientProfileDB *m_ProfileDB;
};
```

Таким образом, с использованием диаграмм компонентов были сформированы шаблоны классов, отражающие структуру модулей системы: интерфейсных форм, контроллера и сущностей данных. Шаблоны кодов могут быть использованы в дальнейшем при непосредственной реализации модулей, расширении функциональных возможностей и интеграции системы.

Создание таких шаблонов позволяет упростить разработку, повысить уровень единообразия и обеспечить строгое соответствие архитектурной модели проектируемой системе.