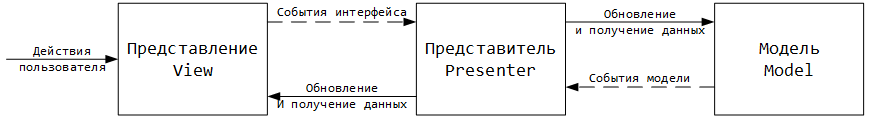
# ПРОЕКТИРОВАНИЕ

## Разработка архитектуры программного продукта

Архитектура программного обеспечения - совокупность важнейших решений об организации программной системы. Процесс проектирования архитектуры программного обеспечения включает в себя сбор требований клиентов, их анализ и создание проекта для компонента программного обеспечения в соответствие с требованиями. Успешная разработка ПО должна обеспечивать баланс неизбежных компромиссов вследствие противоречащих требований; соответствовать принципам проектирования и рекомендованным методам, выработанным со временем; и дополнять современное оборудование, сети и системы управления.

Архитектуру программного обеспечения можно рассматривать как сопоставление между целью компонента ПО и сведениями о реализации в коде. Правильное понимание архитектуры обеспечит оптимальный баланс требований и результатов. Программное обеспечение с хорошо продуманной архитектурой будет выполнять указанные задачи с параметрами исходных требований, одновременно обеспечивая максимально высокую производительность, безопасность, надежность и многие другие факторы.

На самом высоком уровне проект архитектуры должен предоставлять структуру системы, но скрывать детали реализации; охватывать все случаи применения и сценарии; пытаться учитывать требования всех заинтересованных лиц; и удовлетворять настолько, насколько возможно, всем функциональным требованиям и требованиям к качеству.

MVP, как и его родитель, MVC (Model-View-Controller) придуман для удобства разделения бизнес-логики от способа ее отображения.  
  
  
  
На просторах интернета можно встретить целое множество реализаций MVP. По способу доставки данных в представление их можно разделить на 3 категории:  
— Passive View: View содержит минимальную логику отображения примитивных данных (строки, числа), остальным занимается Presenter;  
— Presentation Model: во View могут передаваться не только примитивные данные, но и бизнес-объекты;  
— Supervising Controller: View знает о наличии модели и сам забирает из нее данные.  
  
Далее будет рассматриваться модификация Passive View. Опишем основные черты:  
— интерфейс Представления (IView), который предоставляет некий контракт для отображения данных;  
— Представление — конкретная реализация IView, которая умеет отображать саму себя в конкретном интерфейсе (будь то Windows Forms, WPF или даже консоль) и ничего не знает о том, кто ей управляет. В нашем случае это формы;  
— Модель — предоставляет некоторую бизнес-логику (примеры: доступ к базе данных, репозитории, сервисы). Может быть представлена в виде класса или опять же, интерфейса и реализации;  
— Представитель содержит ссылку на Представление через интерфейс (IView), управляет им, подписывается на его события, производит простую валидацию (проверку) введенных данных; также содержит ссылку на модель или на ее интерфейс, передавая в нее данные из View и запрашивая обновления..

* 1. [**Проектирование структур хранения данных**](#_Toc474748997)

Логическая структура реляционной базы данных определяется совокупностью логически взаимосвязанных реляционных таблиц. Каждая реляционная таблица имеет структуру, определяемую реквизитным составом одного из информационных объектов полученной ИЛМ. Логические связи таблиц соответствуют структурным связям между объектами.

Логическая структура реляционной базы данных, построенная на основе полученной ИЛМ, приведена на рисунке 3.1. На этой схеме реляционные таблицы представлены структурой, определяемой составом и последовательностью полей (атрибутов). Наименования ключевых полей выделены ключом. Логические связи изображены линиями между соответствующими ключами связи.

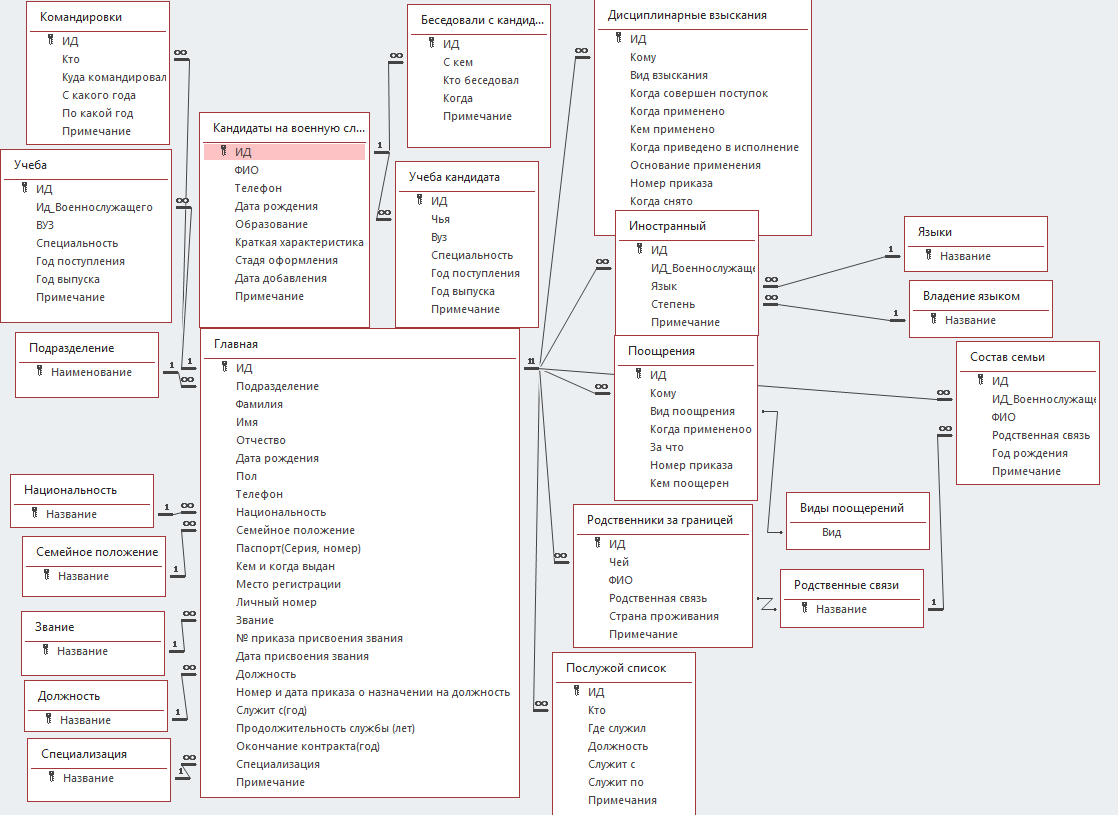


Рисунок 3.1 – Структура базы данных

* 1. [**Описание реализации вариантов использования**](#_Toc474748998)

Моделирование необходимо для понимания системы. Обычно, при этом единственной модели никогда не бывает достаточно. Наоборот, для понимания практически любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки.

Диаграмма вариантов использования показывает, какая функциональность должна быть реализована в системе, основные функции, которые должны быть включены в систему, их окружение и взаимодействие функций с окружением.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Ниже приводится диаграмма вариантов использования, разработанной программы учета и регистрации поступлений материалов. Данная диаграмма отображает возможности и функционал разрабатываемой программы. На рисунке 3.2 представлена диаграмма вариантов использования, на которой указаны все возможные действия, которые может осуществлять пользователь при работе с данной программой.



Рисунок 3.2 - Диаграмма вариантов использования