# 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 [11].

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.

Приложение будет построено на базе архитектуры MVC. Термин ***модель-представление-контроллер (model-view-controller)*** используется с конца 70-х гг. прошлого столетия [12]. Этот шаблон происходит из проекта **Smalltalk**, выполнявшегося в Xerox PARC, где он был задуман как способ организации ранних приложений с графическим пользовательским интерфейсом. Некоторые нюансы первоначального шаблона MVC были связаны с концепциями, специфичными для Smalltalk, такими как экраны и инструменты, но более широкие понятия по-прежнему применимы к приложениям - и особенно хорошо они подходят для веб-приложений.

Взаимодействие с приложением MVC осуществляется в соответствии с естественным циклом действий пользователя и обновлений представления, при котором предполагается, что представление не содержит информации о состоянии. Это прекрасно сочетается с запросами и ответами HTTP, которые лежат в основе веб-приложения.

Более того, инфраструктура MVC принудительно применяет разделение ответственности - *модель предметной области* и *логика контроллера* отделены от пользовательского интерфейса. В веб-приложении это означает, что HTML-разметка хранится отдельно от остальной части приложения, что упрощает и облегчает сопровождение и тестирование.

Появление платформы Ruby on Rails привело к возобновлению всеобщего интереса к MVC, и он остается шаблоном реализации архитектуры MVC. С тех пор появилось много других инфраструктур MVC, и все они демонстрировали преимущества архитектуры MVC – разумеется, это относится и к ASP.NET MVC.

Если оперировать высокоуровневыми понятиями, то архитектурный шаблон MVC означает, что приложение MVC будет разделено, по крайней мере, на три части, которые описаны ниже:

**Модели**, содержащие или представляющие данные, с которыми работают пользователи. Они могут быть простыми моделями представлений, которые только представляют данные, передаваемые между представлениями и контроллерами, или же они могут быть моделями предметной области, которые содержат бизнес-данные, а также операции, преобразования и правила для манипулирования этими данными.

**Представления**, применяемые для визуализации некоторой части модели в виде пользовательского интерфейса.

**Контроллеры**, которые обрабатывают поступающие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

Модели – это определение "вселенной", в которой функционирует ваше приложение. Например, в банковском приложении модель представляет все аспекты банковской деятельности, поддерживаемые приложением, такие как расчетные счета, главная бухгалтерская книга и кредитные лимиты для клиентов, равно как и операции, которые могут использоваться для манипулирования данными в модели, такие как внесение денежных средств и списание со счетов. Модель отвечает также за сохранение общего состояния и целостности данных – например, удостоверяясь, что все транзакции внесены в главную книгу, а клиент не снимает со счета больше денежных средств, чем имеет на то право, или больше, чем имеется в распоряжении самого банка.

Модели определяются также и тем, за что они не отвечают: модели не имеют дела с визуализацией пользовательских интерфейсов или обработкой запросов, т.к. это ответственность представлений и контроллеров.

Представления содержат логику, необходимую для отображения элементов модели пользователю – и ничего более. Они не имеют никаких прямых сведений о модели и не обмениваются данными непосредственно с ней. Контроллеры являются шлюзом между представлениями и моделью – запросы поступают от клиента и обслуживаются контроллером, который выбирает подходящее представление для отображения пользователю и, если требуется, соответствующую операцию, которая должна быть выполнена над моделью.

Каждая часть архитектуры MVC является четко определенной и самодостаточной – это то, что называется *разделением ответственности*. Логика, которая манипулирует данными в модели, содержится только в модели, логика, отображающая данные – только в представлении, а код, который обрабатывает запросы и ввод пользователей – только в контроллере. При четком разграничении всех частей приложение будет легче сопровождать и расширять в течение его срока существования, независимо от того, насколько большим оно станет.

* 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 [13].

Логическая структура реляционной базы данных, построенная на основе полученной пунктом ранее концептуальной модели, 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.

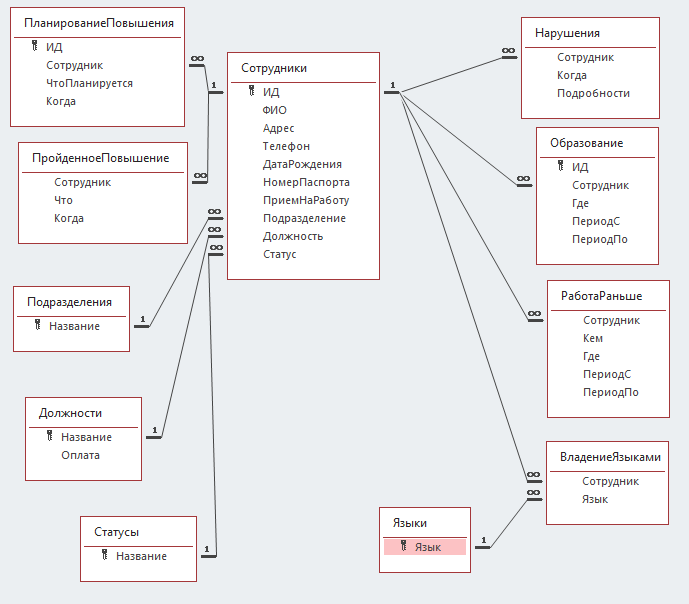


Рисунок 3.1 – Структура базы данных

* 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 [14].

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.

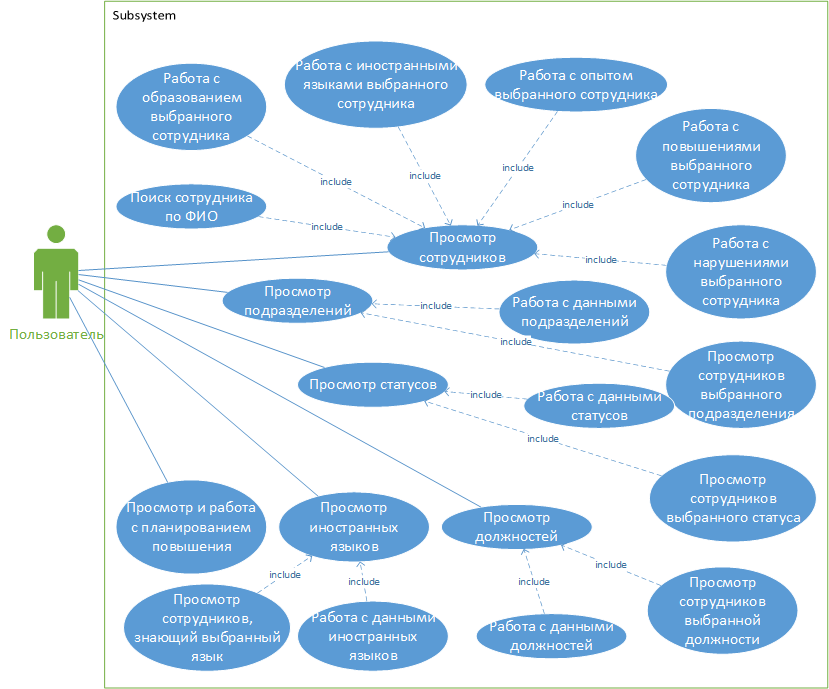


Рисунок 3.2 - Диаграмма вариантов использования