**3 Разработка программно-аппаратного комплекса исследования динамики заучивания**

**3.1 Разработка структуры программы**

Модель-представление-контроллер (MVC — Model-view-controller) — это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам. Впервые паттерн MVC появился в языке smalltalk [12]. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название [13].

Под моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму модель [12].

Модель обладает следующими признаками:

* Модель — это бизнес-логика приложения;
* Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
* Для некоторых проектов модель — это просто слой данных (dao, база данных, xml-файл);
* Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения[13].

В обязанности представления входит отображение данных полученных от модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным [14].

Примеры представления: html-страница, wpf форма, windows form.  
Представление обладает следующими признаками:

* В представлении реализуется отображение данных, которые получаются от модели любым способом;
* В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику [15][16].

Наиболее распространенные виды mvc-паттерна, это:

* Model-view-controller [рис. 3.2.1]
* Model-view-presenter [рис. 3.2.2]
* Model-view-view model [рис. 3.2.3]

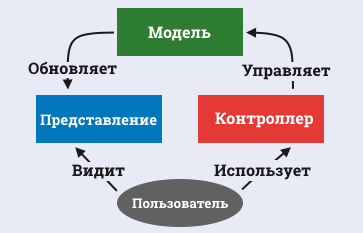


Рисунок 3.2.1 – Шаблон проектирования «Модель-представление-контроллер»

Основная идея этого паттерна в том, что и контроллер, и представление зависят от модели, но модель никак не зависит от этих двух компонент.  
**Признаки контроллера:**

* Контроллер определяет, какие представление должно быть отображено в данный момент;
* События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление.
* Возможно несколько представлений только для одного контроллера [17].

При реализации контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя модель, посредством вызова соответствующего метода. После изменения модель использует событие о том, что она изменилась, и все подписанные на это события представления, получив его, обращаются к модели за обновленными данными, после чего их и отображают. Пример использования: **mvc asp.net** [17].

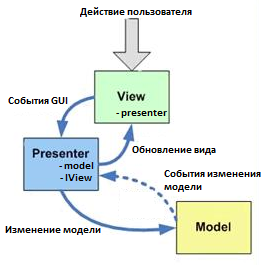


Рисунок 3.2.2 – Шаблон проектирования «Модель-представление-представитель»

Подход «Модель-представление-представитель» позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель [16].

**Признаки презентера:**

* Двухсторонняя коммуникация с представлением;
* Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
* Презентер взаимодействует с view путем использования специального интерфейса, реализованного представлением;
* Один экземпляр презентера связан с одним отображением [17][18].

Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, **iview**.showerrormessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе [18].

Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. cоздания других представлений). Пример использования шаблона: технология**windows forms** [17].

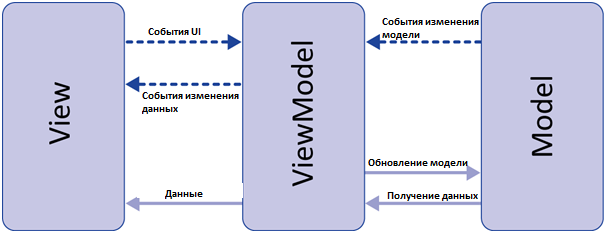


Рисунок 3.2.3 – Шаблон проектирования «Модель-представление-представление»

Подход позволяет связывать элементы представления со свойствами и событиями view-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя[18].

**Признаки view-модели:**

* Двухсторонняя коммуникация с представлением;
* View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами view-модели / модели
* View-модель не имеет ссылки на интерфейс представления (iview). Изменение состояния view-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (bindings)
* Один экземпляр view-модели связан с одним отображением [18].

При использовании этого паттерна, представление не реализует соответствующий интерфейс (iview). Представление должно иметь ссылку на источник данных (datacontex), которым в данном случае является view-модель. Элементы представления связаны (bind) с соответствующими свойствами и событиями view-модели. В свою очередь, view-модель реализует специальный интерфейс, который используется для автоматического обновления элементов представления. Примером такого интерфейса в wpf может быть inotifypropertychanged [18].

Таки образом в основу системы была положена динамически подгружаемая библиотека, реализующая бизнес-модели. С помощью классов слоя доступа к данным устанавливается связь между бизнес-логикой программы и данными. Бизнес модели представляют собой слой Model паттерна MVP [14].

Бизнес модели должны быть трансформированы в модели для представлений. Это позволит разъединить слой представлений от слоя бизнес-логики и слоя доступа к данным. Так же в эти модели можно добавить функционал, не нужный для бизнес-слоя, как например валидация. Данный слой будет «представлять» бизнес-модели для графического интерфейса (presenter) [16].

Слой представлений представляет собой набор форм на платформе Windows Presentation Forms. Данная технология позволит стилизовать пользовательский интерфейс в последствии. Представления будут частью программно-аппаратного комплекса, непосредственно взаимодействующими с пользователем и реализующими свойства дружественности, гибкости и простоты интерфейса пользователя.

Разделение программы по слоям в соответствии с подзадачами, выполняемыми программно-аппаратным комплексом в ходе работы (слой доступа к данным, бизнес-логика, представление) позволяет тестировать программу по частям, что значительно облегчит поиск и устранение дефектов. Так же наличие отдельного слоя доступа к данным позволяет при необходимости разработать другой модуль и использовать данный комплекс с другими источниками данных.

Возможность конфигурирования соединения с базой данных позволяет устанавливать соединение как с локальной, так и с удаленной базой данных. Это способствует развертыванию приложения на нескольких компьютерах и создавать распределенную систему.

В ходе дальнейшего развития системы можно использовать возможности фреймворка Windows Presentation Forms для изменения внешнего вида графического интерфейса пользователя, его интернационализации, а так же развить систему для улучшения возможности выполнять опыты на различных языках.

**3.2 Разработка структуры базы данных**

Реляционная база данных представляет собой набор таблиц для хранения абстрактных объектов (сущностей). Таблицы состоят из колонок и строк (кортежей). Внутри таблиц могут быть определены ограничения, между таблицами существуют отношения. Для описания модели хранимых данных применяются диаграммы баз данных [рис. 3.1.1]. При помощи языка запросов можно выполнять запросы, которые возвращают наборы данных, получаемых из одной или нескольких таблиц [9].

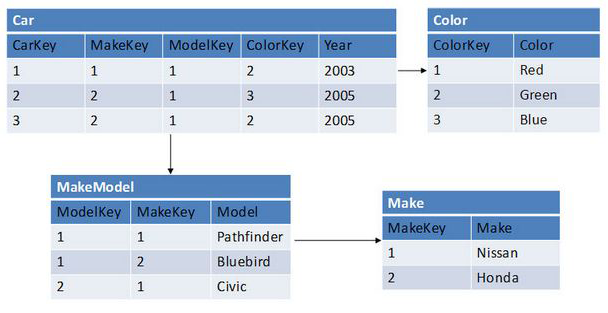


Рисунок 3.1.1 – Модель типичной реляционной базы данных

Структурированный Язык запросов (Structured Query Language — SQL) — это стандартный язык доступа к БД, таким как SQL Server, Oracle, MySQL, Sybase и Access. Знание SQL необходимо всем, кто хотел бы хранить и извлекать данные из БД [9].

SQL предназначен для выполнения запросов. Кроме того, в SQL входит синтаксис для обновления, вставки и уничтожения данных. Этот синтаксис вместе с командами обновления формирует язык управления данными (DML):

* SELECT — извлекает данные из таблицы БД
* UPDATE — обновляет данные в таблице БД
* DELETE — уничтожает данные в таблице БД
* INSERT INTO — вставляет новые данные в таблицу БД [11].

SQL Data Definition Language (DDL) — язык определения данных. DDL является частью SQL, которая управляет созданием и удалением таблиц в БД, Кроме того, с помощью DDL мы можем назначать индексы (ключевые слова), налаживать взаимосвязи между таблицами и накладывать ограничения на таблицы БД.

Важнейшими командами DDL являются следующие команды:

* CREATE TABLE — создание новой таблицы
* ALTER TABLE — изменение существующей таблицы
* DROP TABLE — удаление таблицы
* CREATE INDEX — создание индекса
* DROP INDEX — удаление индекса [11].

В рамках одного запроса данные получаются из нескольких таблиц путем их соединения (JOIN), чаще всего для соединения используются те же колонки, которые определяют отношения между таблицами. Нормализация — это процесс структурирования модели данных, обеспечивающий связность и отсутствие избыточности в данных [10].

Доступ к реляционным базам данных осуществляется через реляционные системы управления базами данных (РСУБД). Почти все системы баз данных, которые мы используем, являются реляционными, такие как Oracle, SQL Server, MySQL, Sybase, DB2, TeraData и так далее [11].

На протяжении всего существования реляционных БД они постоянно предлагали наилучшую смесь простоты, устойчивости, гибкости, производительности, масштабируемости и совместимости в сфере управлении данными [12].

Однако чтобы обеспечить все эти особенности, реляционные хранилища невероятно сложны внутри. Например, простой SELECT запрос может иметь сотни потенциальных путей выполнения, которые оптимизатор оценит непосредственно во время выполнения запроса. Все это скрыто от пользователей, однако внутри РСУБД создает план выполнения, основывающийся на вещах вроде алгоритмов оценки стоимости и наилучшим образом отвечающий запросу [11].

Одним из важнейших путей достижения высокой производительности SQL Server является использование индексов. Индекс ускоряет процесс запроса, предоставляя быстрый доступ к строкам данных в таблице, аналогично тому, как указатель в книге помогает вам быстро найти необходимую информацию [10].

Индексы создаются для столбцов таблиц и представлений. Индексы предоставляют путь для быстрого поиска данных на основе значений в этих столбцах. Например, если создаётся индекс по первичному ключу, а затем будет производится поиск строки с данными, используя значения первичного ключа, то SQL Server сначала найдет значение индекса, а затем использует индекс для быстрого нахождения всей строки с данными. Без индекса будет выполнен полный просмотр (сканирование) всех строк таблицы, что может оказать значительное влияние на производительность [10].

Создать индекс на большинстве столбцов таблицы или представления. Исключением, преимущественно, являются столбцы с типами данных для *хранения больших объектов (LOB), таких как image, text или varchar(max).* Можно создать индексы на столбцах, предназначенных для хранения данных в формате *XML*, но эти индексы устроены иначе чем стандартные [12].

Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — *сбалансированного дерева*. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев, в нижней части [рис. 3.1.2].

При формировании запроса на индексированный столбец, подсистема запросов начинает идти сверху от корневого узла и постепенно двигается вниз через промежуточные узлы, при этом каждый слой промежуточного уровня содержит более детальную информацию о данных. Подсистема запросов продолжает двигаться по узлам индекса до тех пор, пока не достигнет нижнего уровня с листьями индекса [12].

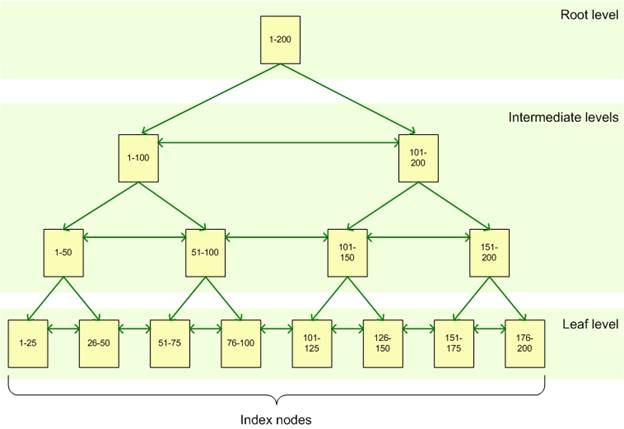


Рисунок 3.1.2 – Структура индекса базы данных

К примеру, если ищется значение 123 в индексированном столбе, то подсистема запросов сначала на корневом уровне определит страницу на первом промежуточном (intermediate) уровне. В данном случае первой страница указывает на значение от 1 до 100, а вторая от 101 до 200, таким образом подсистема запросов обратится ко второй странице этого промежуточного уровня. Далее будет выяснено, что следует обратиться к третьей странице следующего промежуточного уровня. Отсюда подсистема запросов прочитает на нижнем уровне значение самого индекса [рис. 3.1.2].

При анализе предметной области были выделены данные, подлежащие хранению в базе данных:

* Информация о пользователе
* Информация об опытах
* Информация о стимулах
* Результаты выполнения опытов [рис. 3.1.3].

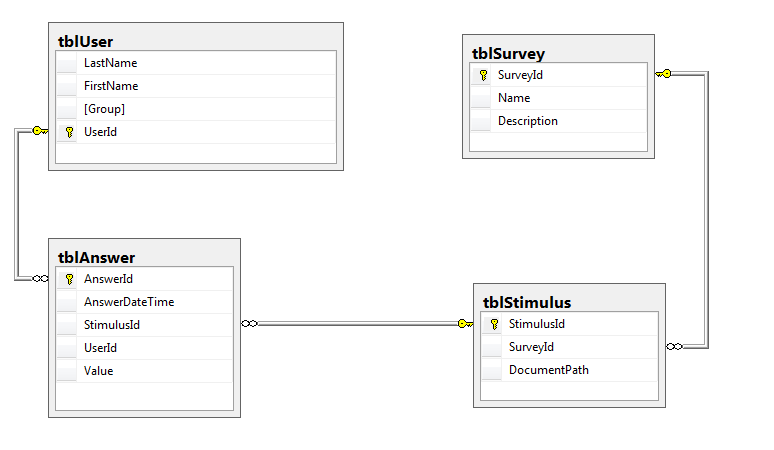


Рисунок 3.1.3 Основные сущности базы данных

Дополнительно было принято решение сохранить в базе данных настройки опытов. Настройки представляют собой абстрактную сущность, хранящую информацию о времени предъявления стимула, последовательности предъявления стимула, а также имеющую связь (ссылку) на набор стимулов. В базу данных был введен новая сущность [рис. 3.1.4]

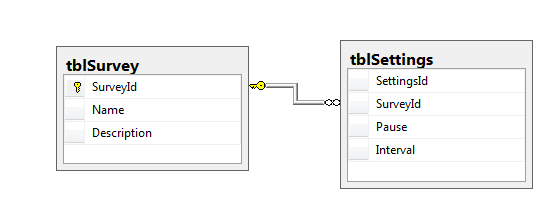


Рисунок 3.1.4 – Схема хранения настроек в базе данных.

Выполненный проект базы данных отражает основные свойства предметной области и при необходимости может быть модифицирован на этапе разработки программного модуля.

**3.3 Тестирование программы**

Тестирование программного обеспечения (Software Testing) - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004] В более широком смысле, тестирование - это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Верификация (Verification) - это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа [IEEE]. Т.е. выполняются ли наши цели, сроки, задачи по разработке проекта, определенные в начале текущей фазы.

Валидация (Validation) - это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе [BS7925-1].

План тестирования (Test Plan) - это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Тест дизайн (Test Design) - это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

Тестовый случай (Test Case) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Баг/дефект репорт (Bug Report) - это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Тестовое покрытие (Test Coverage) - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

Детализация тест кейсов (Test Case Specification) - это уровень детализации описания тестовых шагов и требуемого результата, при котором обеспечивается разумное соотношение времени прохождения к тестовому покрытию.

Время прохождения тест кейса (Test Case Pass Time) - это время от начала прохождения шагов тест кейса до получения результата теста.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (компонентном, интеграционном, системном, приемочном). Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

Тестирование функциональности может проводиться в двух аспектах:

— требования;

— бизнес-процессы.

Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу для дизайна тестовых случаев (Test Cases). В этом случае необходимо сделать список того, что будет тестироваться, а что нет, приоритезировать требования на основе рисков (если это не сделано в документе с требованиями), а на основе этого приоритезировать тестовые сценарии (test cases). Это позволит сфокусироваться и не упустить при тестировании наиболее важный функционал.

Тестирование в перспективе «бизнес-процессы» использует знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы. В этой перспективе тестовые сценарии (test scripts), как правило, основываются на случаях использования системы (use cases). Достаточно распространенной является автоматизация функционального тестирования.

Преимущества функционального тестирования:

— имитирует фактическое использование системы.

Недостатки функционального тестирования:

— возможность упущения логических ошибок в программном обеспечении;

— вероятность избыточного тестирования.

Тестирование безопасности - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Общая стратегия безопасности основывается на трех основных принципах:

— конфиденциальность;

— целостность;

— доступность;

— конфиденциальность.

Конфиденциальность - это сокрытие определенных ресурсов или информации. Под конфиденциальностью можно понимать ограничение доступа к ресурсу некоторой категории пользователей, или другими словами, при каких условиях пользователь авторизован получить доступ к данному ресурсу.

Существует два основных критерия при определении понятия целостности:

— Доверие. Ожидается, что ресурс будет изменен только соответствующим способом определенной группой пользователей.

— Повреждение и восстановление. В случае когда данные повреждаются или неправильно меняются авторизованным или не авторизованным пользователем, вы должны определить на сколько важной является процедура восстановления данных.

Доступность представляет собой требования о том, что ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству. Как правило, чем более критичен ресурс тем выше уровень доступности должен быть.

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

— измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;

— определение количества пользователей, одновременно работающих с приложением;

— определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций);

— исследование производительности на высоких, предельных, стрессовых нагрузках.

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

— измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;

— может производиться определение количества пользователей, одновременно работающих с приложением.

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.