Министерство образования Тульской области  
Государственное профессиональное образовательное учреждение   
Тульской области   
«Донской колледж информационных технологий»

РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ПО КУРСУ   
«АНГЛИЙСКИЙ ЯЗЫК»

Выпускная квалификационная работа

|  |  |
| --- | --- |
| Студент группы 4-П-1 | Гавришин Евгений Юрьевич |
| Руководитель | Качанов Алексей Владимирович |
| Рецензент | Горбунова Анастасия Сергеевна |

Донской, 2018

СОДЕРЖАНИЕ

Введение3

1. Техническое задание5
   1. Введение5
   2. Основание для разработки5
   3. Назначение разработки5
   4. Требования к программному продукту5
   5. Требования к составу и параметрам технических средств6
   6. Требования к программной документации7
2. Построение и проектирование моделей8

3 Реализация16

3.1 Обоснование выбора средств разработки16

* 1. Описание программных модулей18

3.3 Создание приложения20

3.4 Паттерн проектирования24

4 Модульное тестирование программного продукта и анализ кода28

5 Руководство пользователя31

Заключение38

Список использованных источников39

Приложение А. Листинг управляющего кода40

Приложение Б. Окна47

ВВЕДЕНИЕ

В настоящее время востребованность знания английского языка достигла крайне высокого уровня в самых разных сферах человеческой деятельности. Инженеры, журналисты, экономисты и тем более специалисты сферы IT, а также многие другие специалисты сейчас столкнулись с проблемой – чтобы им оставаться востребованными на рынке труда, им необходимо знать английский язык хотя бы на базовом уровне, читать и переводить со словарём. Эта проблема появилась не просто так. Работодателю выгоднее иметь работников, знающих английский язык, поскольку практически все новинки, разработки, методики и технологии приходят в нашу страну из-за рубежа, а английский язык является международным языком. Также нельзя не отметить наличие у многих людей желания изучать английский язык для своего собственного развития, чтобы языковой барьер не мешал путешествовать, быть в контакте со всем миром, иметь иностранных друзей.

Изучая английский язык в школе, колледже или университете обучающиеся делают это неохотно. У них есть желание, но нет мотивации. Зачастую все выглядит очень скучным, много материала усваивается одной огромной порцией один или два раза в неделю, а через продолжительное время изученное и вовсе исчезает из памяти, поскольку нет практики. И в если ситуация в школе еще не так плоха, то в колледже или университете обычно на английский язык отводится мало часов, занятия обеднены практикой и развитием активных навыков, объяснения преподавателя содержат очень сложные для восприятия термины, а материалы быстро устаревают. Также нельзя не отметить то, что класс или группа состоят обычно из десяти человек и больше, а преподаватель не всегда может физически передать информацию доступно и индивидуально каждому. Людям, которые заинтересованы в изучении нужен инструмент, который позволяет автоматически четко контролировать процесс обучения, подстраиваться под возможности, учитывать индивидуальные особенности и быть доступным в любую минуту.

Исходя из вышенаписанного можно прийти к выводу, что для решения данной проблемы следует объединить силы профессионалов: преподавателей английского языка и разработчиков программного обеспечения и разработать программный продукт, который позволит людям имея персональный компьютер, ноутбук или планшет решить их проблему – изучить английский язык таким образом, чтобы им это было интересно, сложные вещи объяснялись как можно проще, сеансы занятий проводились регулярно и в спокойной обстановке, знания проверялись интерактивно, с использованием материалов, интересных пользователю, а ошибки не порицались, а были призывом к дальнейшему развитию.

Целью выпускной квалификационной работы является разработка Windows приложения на языке программирования C# с платформой .NET Framework 4.6 для изучения английского языка.

Задачи выпускной квалификационной работы:

* изучение предметной области;
* разработка Windows приложения;
* разработка технического задания;
* работа с диаграммами классов;
* применение объектно-ориентированного подхода к программированию;
* проектирование приложения в соответствии с Model-View-Presenter (MVP) и Singleton паттернами;
* написание кода с высоким индексом удобства поддержки;
* разработка современного интерфейса пользователя;
* использование модульных тестов для тестирования модели;
* разработка инсталлятора;

Объектом исследования выпускной квалификационной работы является методика применения программных приложений в процессе обучения.

Предметом исследования является применение программного обеспечения в обучении английскому языку.

1 Техническое задание

1.1 Введение

Настоящее техническое задание распространяется на разработку прикладного программного обеспечения (далее ПО), позволяющего изучать английский язык. Наименование программы – «EnglishLab».

Программа предназначена к применению всеми желающими, а также может оказать помощь при изучении учениками, студентами дисциплины «Иностранный язык» в школе, колледже или университете.

1.2 Основание для разработки

Основанием для проведения разработки является распорядительный акт государственного профессионального образовательного учреждения Тульской области «Донской колледж информационных технологий» (далее ГПОУ ТО «ДКИТ») о подготовке и процедуре защиты выпускных квалификационных работ студентов выпускных курсов по специальности 09.02.03 «Программирование в компьютерных системах».

1.3 Назначение разработки

Назначением программы является предоставление пользователю возможности изучать английский язык самостоятельно, закреплять полученные навыки, контролировать процесс обучения, пополнять словарный запас, изучать материалы на интересующие темы.

1.4 Требования к программному продукту

Программа должна обеспечить возможность выполнения следующих функций:

* тестирование нового пользователя для определения его уровня знаний в различных областях английского языка;
* управление готовыми наборами слов для изучения (добавление к изучению, удаление);
* управление пользовательским набором слов (добавление в него новых слов, их удаление);
* вычисление коэффициента заученности каждого слова во всех наборах слов;
* режим изучения набора слов «Словарные карточки»;
* режим изучения набора слов «Слово-перевод»
* режим изучения набора слов «Перевод-слово»;
* тестирование на усвоение пройденных грамматических тем;
* режим чтения материалов;
* сохранение всей информации о прогрессе обучения в файл;
* возможность возобновлять прогресс из файла;
* автозагрузка программы и напоминание пользователю о занятиях;
* сбор и предоставление пользователю статистики;

Интерфейс должен иметь современный и лаконичный дизайн (например, Flat дизайн), быть интуитивно понятным, дружественным пользователю приложения и не нуждаться в подробном руководстве по его эксплуатации.

1.5 Требования к составу и параметрам технических средств

Программа должна работать на устройствах, имеющих x86/x64 совместимую операционную систему семейства Microsoft Windows. Минимальная версия операционной системы, на которой возможна работа программы – Windows 7. Рекомендуемая – Windows 10.

На устройстве должна быть установлена как минимум 4.0 версия программной платформы .NET Framework. Рекомендуется: .NET Framework 4.6.

Минимальное разрешение дисплея 1024 x 768. Остальные системные требования данной программы к аппаратной части устройства совпадают с системными требованиями операционной системы Microsoft Windows любой редакции, начиная с Windows 7, а именно:

Процессор: минимум 1 ГГц.

ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем).

Место на жестком диске: минимум 20 ГБ.

Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0.

1.6 Требования к программной документации

Комментарии — пояснения к исходному тексту программы, находящиеся непосредственно внутри комментируемого кода. Синтаксис комментариев определяется языком программирования. С точки зрения компилятора или интерпретатора, комментарии — часть текста программы, не влияющая на её семантику. Комментарии не оказывают никакого влияния на результат компиляции программы или её интерпретацию. Помимо исходных текстов программ, комментарии также применяются в языках разметки и языках описания.

Разрабатываемые программные модули, должны быть самодокументируемы, то есть пространства имён, типы, поля, методы, переменные должны иметь понятные человеку идентификаторы и не нуждаться в дополнительных комментариях, либо содержать короткие однострочные комментарии, если они необходимы для пояснения работы алгоритма (части кода) решения поставленной задачи.

Разработанные самостоятельно dll-библиотеки должны иметь XML комментарии, но только в тех местах, где это необходимо.

2 Построение и проектирование моделей

При объектном подходе к программированию модели разрабатываемой системы основываются на предметах и явлениях окружающего мира.

Модель — упрощенное представление реальности. С точки зрения программирования модель — это чертеж системы.

Моделирование необходимо для решения следующих задач:

* визуализации системы;
* определения ее структуры и поведения;
* получения шаблона, позволяющего затем сконструировать систему;
* документирования принимаемых решений, используя полученные модели.

Для решения этих задач при описании поведения проектируемого программного обеспечения в настоящее время используется UML (Unified Modeling Language) — унифицированный язык моделирования.

В основе объектного подхода к разработке программного обеспечения лежит объектная декомпозиция, т. е. представление разрабатываемого программного обеспечения в виде совокупности объектов, в процессе взаимодействия которых через передачу сообщений происходит выполнение требуемых функций.

Для создания моделей анализа и проектирования объектно-ориентированных программных систем используют языки визуального моделирования, самым популярным из которых на сегодняшний день является UML. Схемы отдельных модулей в виде диаграмм классов представлены на рисунках 2.1 – 2.11.

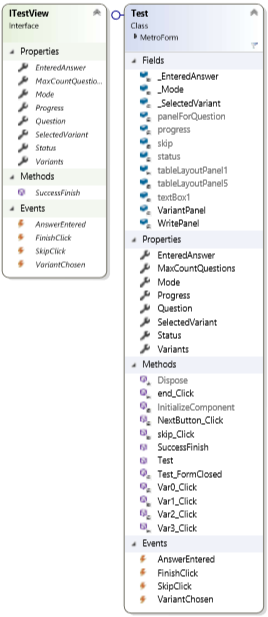


Рисунок 2.1 – Диаграмма класса Test

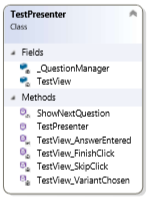


Рисунок 2.2 – Диаграмма класса TestPresenter

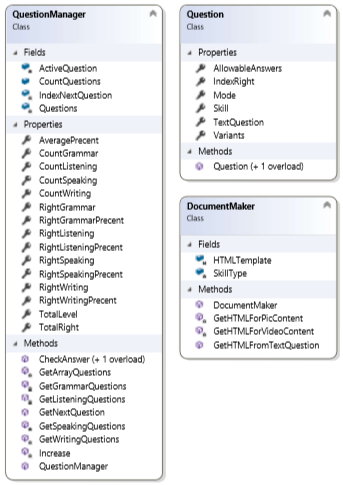


Рисунок 2.3 – Диаграмма классов QuestionManager, Question и DocumentMaker.

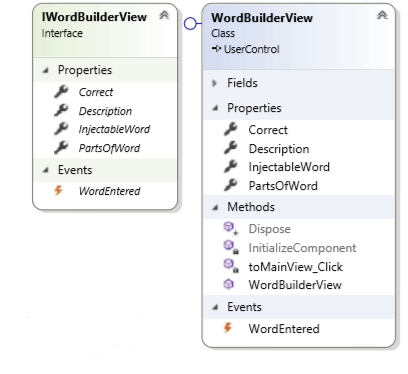


Рисунок 2.4 – Диаграмма класса WordBuildView.

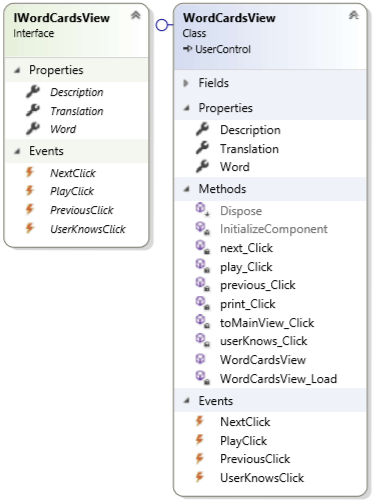


Рисунок 2.5 – Диаграмма класса WordCardsView.

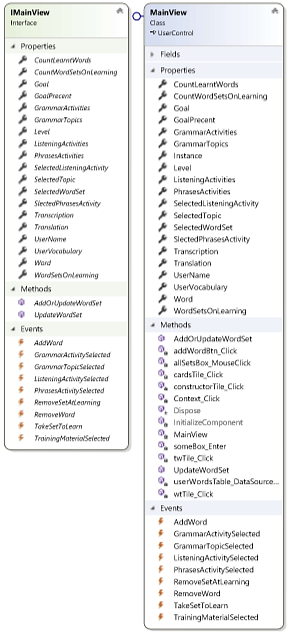


Рисунок 2.6 – Диаграмма класса MainView.

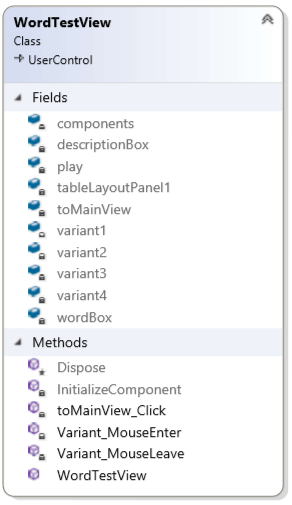


Рисунок 2.7 – Диаграмма класса WordTestView.

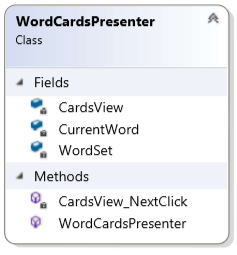


Рисунок 2.8 – Диаграмма класса WordCardsPresenter.

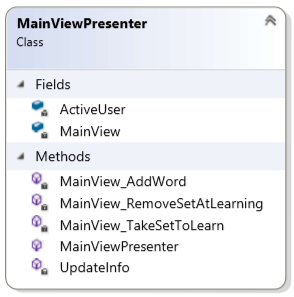


Рисунок 2.9 – Диаграмма класса MainViewPresenter.

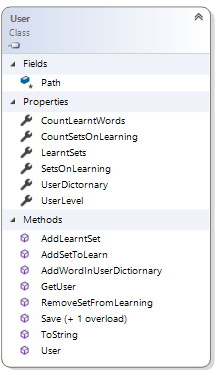


Рисунок 2.10 – Диаграмма класса User.

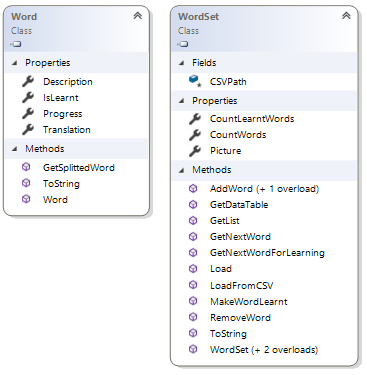


Рисунок 2.11 – Диаграмма классов Word и WordSet.

3 Реализация

3.1 Обоснование выбора средств разработки

Интегрированная среда разработки (IDE) – это комплексное средство, включающее всё необходимое программисту для создания программного обеспечения. Чёткой дефиниции для этого термина не существует: никто не скажет вам про тот или иной программный продукт, что эта среда разработки менее интегрирована, а та - более. Тем не менее, существует некоторый набор компонентов, которые должны присутствовать в интегрированных средах разработки. Во-первых, это компилятор или интерпретатор, во-вторых - редактор исходного кода программ (обязательно хотя бы с поддержкой подсветки синтаксиса того языка программирования, для которого предназначена среда), а в-третьих - отладчик. Отладчик - это даже более существенная часть интегрированной среды разработки, чем компилятор или интерпретатор, поскольку нередко именно отладка программы становится самым сложным и дорогостоящим этапом её создания.

Конечно, современные интегрированные среды разработки предлагают программистам гораздо больше возможностей, чем входят в описанный выше необходимый минимум. Например, многие современные IDE являются визуальными - они позволяют создавать интерфейс программы с помощью мышки, точно в таком виде, в каком он предстанет потом пользователю. IDE, не являющиеся визуальными, требуют от программиста писать специальный код, ответственный за создание пользовательского интерфейса программы.

В зависимости от того, для каких платформ можно писать программы и на каких платформах работает сама IDE, среды разработки разделяются на кроссплатформенные (поддерживают работу с разными платформами) или платформенно-зависимые (те, которые работают только с одной платформой). Классический пример кроссплатформенной среды разработки – Android Studio, платформенно-зависимой – Visual Studio.

В зависимости от количества поддерживаемых языков программирования, среды могут быть многоязычными или одноязычными.

Список популярных сред разработки велик, и все значимые продукты этого класса динамично развиваются в сторону всё большего удобства для разработчиков.

На сегодняшний день для разработки на платформе .NET пользуются популярностью две среды разработки: JetBrains Rider и Microsoft Visual Studio.

Компания JetBrains основана выпускниками Санкт-Петербургского государственного университета и специализируется на создании инструментов для разработчиков. Штаб-квартира находится в Праге. При этом все продукты разрабатываются в основном в Санкт-Петербурге и Мюнхене русскими программистами. В питерском офисе трудится около 400 человек. Есть также офисы в Москве и Бостоне. Среда разработки JetBrains Rider (Рисунок 3.1.1) является удобной для .NET разработчика, но в то же время эта среда является платной.

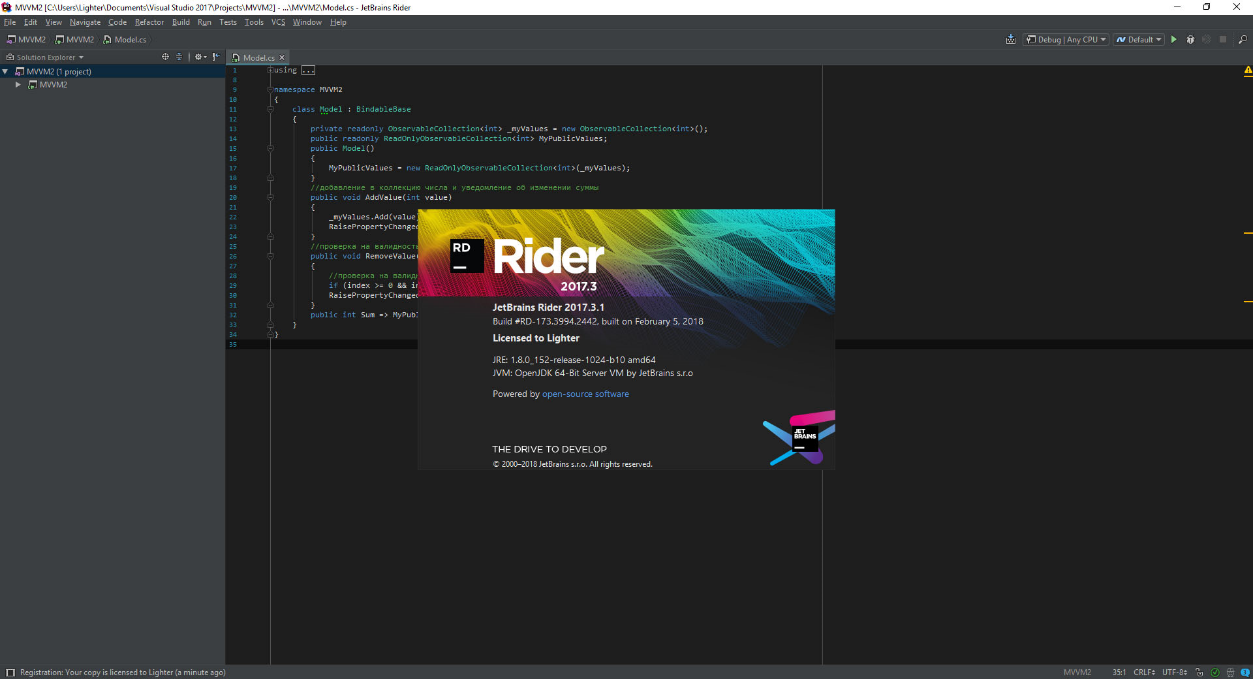


Рисунок 3.1.1 - Среда разработки JetBrains Rider

Среда разработки Visual Studio Community (Рисунок 3.1.2) разработана всемирно известной корпорацией Microsoft, а также она полностью бесплатна для индивидуальных разработчиков и при этом она полнофункциональна.

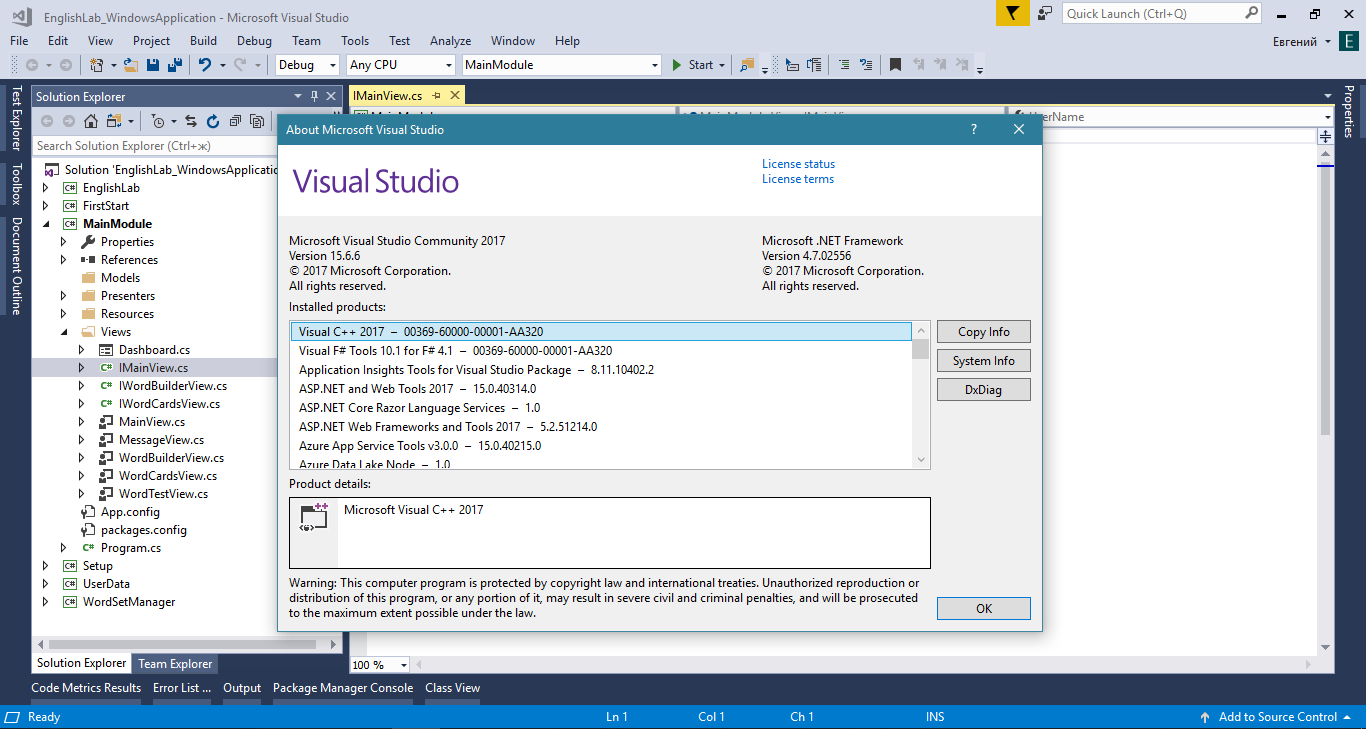


Рисунок 3.1.2 - Среда разработки Visual Studio Community.

Проанализировав информацию об этих двух средах разработки, было принято решение, что наиболее подходящая среда разработки это Microsoft Visual Studio 2017 редакции Community.

3.2 Описание программных модулей

При создании в Visual Studio приложения, веб-сайта, веб-приложения, скрипта, подключаемого модуля и т. д. вы начинаете с проекта. С логической точки зрения проект содержит все файлы исходного кода, значки, изображения, файлы данных и прочие элементы, которые будут скомпилированы в исполняемую программу или веб-сайт, а также все остальное, что необходимо для выполнения компиляции. Проект также содержит все параметры компилятора и другие файлы конфигурации, которые могут потребоваться разным службам или компонентам, с которыми программа будет взаимодействовать.

С точки зрения логики и файловой системы проект содержится в рамках решения, которое может содержать один или несколько проектов вместе с информацией о сборке, параметрами окна Visual Studio и любыми прочими файлами, которые не связаны с каким-либо проектом. Фактически решение является текстовым файлом в собственном уникальном формате; его обычно не изменяют вручную.

То есть можно сказать, что решение является неким контейнером, включающим в себя множество проектов.

Модуль — функционально законченный фрагмент программы. Во многих языках (но далеко не обязательно) оформляется в виде отдельного файла с исходным кодом или поименованной непрерывной её части. Некоторые языки предусматривают объединение модулей в библиотеки или отдельные проекты.

Каждый проект в решении является модулем приложения, который содержит программный код.

Структура решения представлена на рисунке 3.2.1.

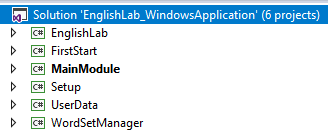


Рисунок 3.2.1 – Структура решения.

Модуль «EnglishLab» управляет запуском других модулей в зависимости от того, запускает пользователь приложение впервые или уже является пользователем.

Модуль «FirstStart» содержит в себе код программы, предназначенной для первого запуска. При первом запуске собирается информация о пользователе, а также может производиться тестирование.

Модуль «MainModule» является главным. Он содержит в себе главную программу, которая позволяет пользователю заниматься тренировками, получать статистику и читать материалы.

Модуль «Setup» представляет собой программу установки приложения на компьютер пользователя.

Модуль «UserData» представляет модель данных пользователя.

Модуль «WordSetManager» содержит логику работы приложения, как например алгоритмы работы тренажеров.

3.3 Создание приложения

Разработка приложения начинается с создания проекта типа Windows Forms App (Рисунок 3.3.1).

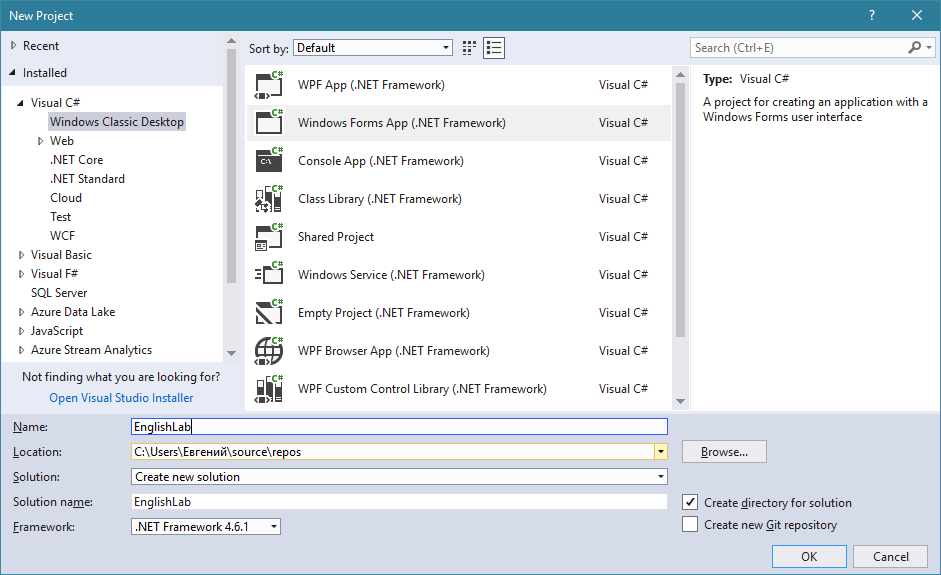


Рисунок 3.3.1 – Окно создания проекта.

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке MFC, изначально написанной на языке C++. С другой стороны, Windows Forms не предлагает парадигму, сравнимую с MVC. Для исправления этой ситуации и реализации данной функциональности в Windows Forms существуют сторонние библиотеки. Одной из наиболее используемых подобных библиотек является User Interface Process Application Block, выпущенная специальной группой Microsoft, занимающейся примерами и рекомендациями, для бесплатного скачивания. Эта библиотека также содержит исходный код и обучающие примеры для ускорения обучения.

Внутри .NET Framework, Windows Forms реализуется в рамках пространства имён System.Windows.Forms.

После создания проекта перед разработчиком открывается окно, изображенное на рисунке 3.3.2.

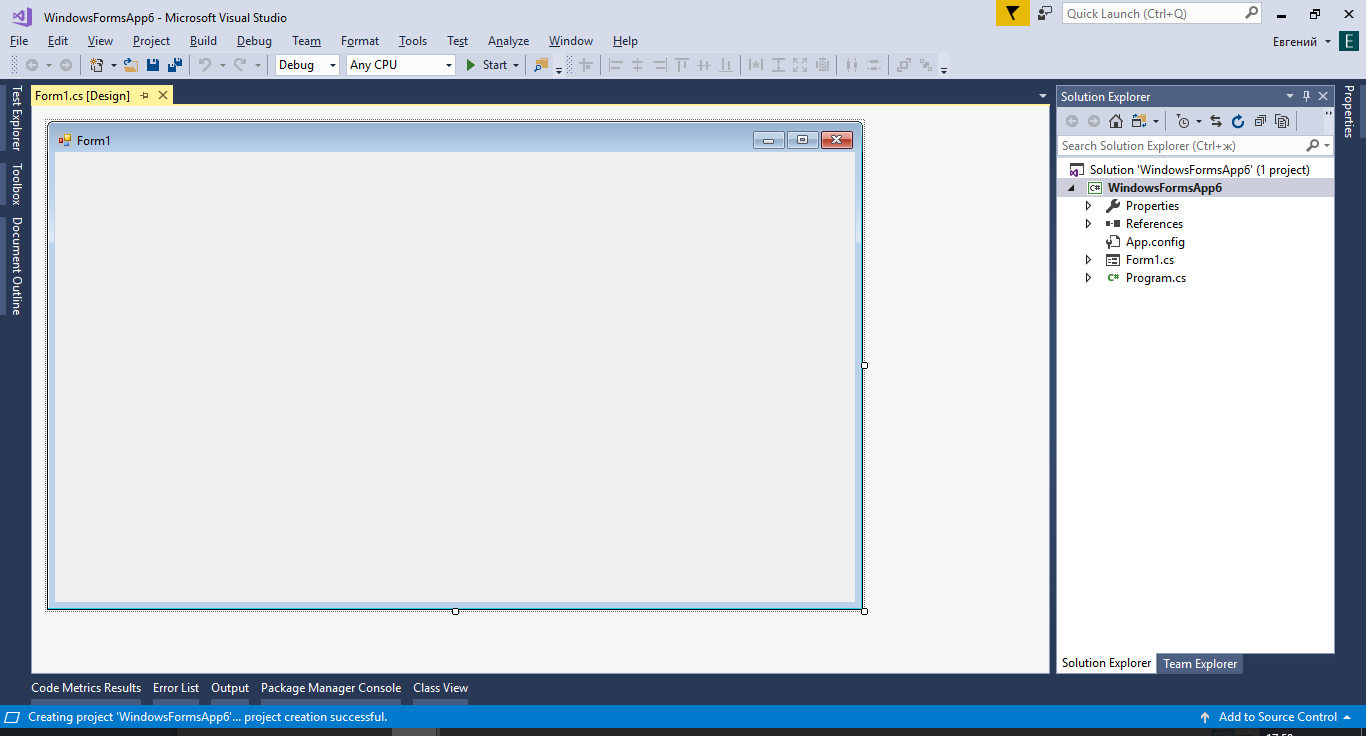


Рисунок 3.3.2 – Конструктор форм.

При помощи конструктора форм создаются различные представления для приложения. На рисунках 3.3.3, 3.3.4, 3.3.5 изображена работа по проектированию представлений.

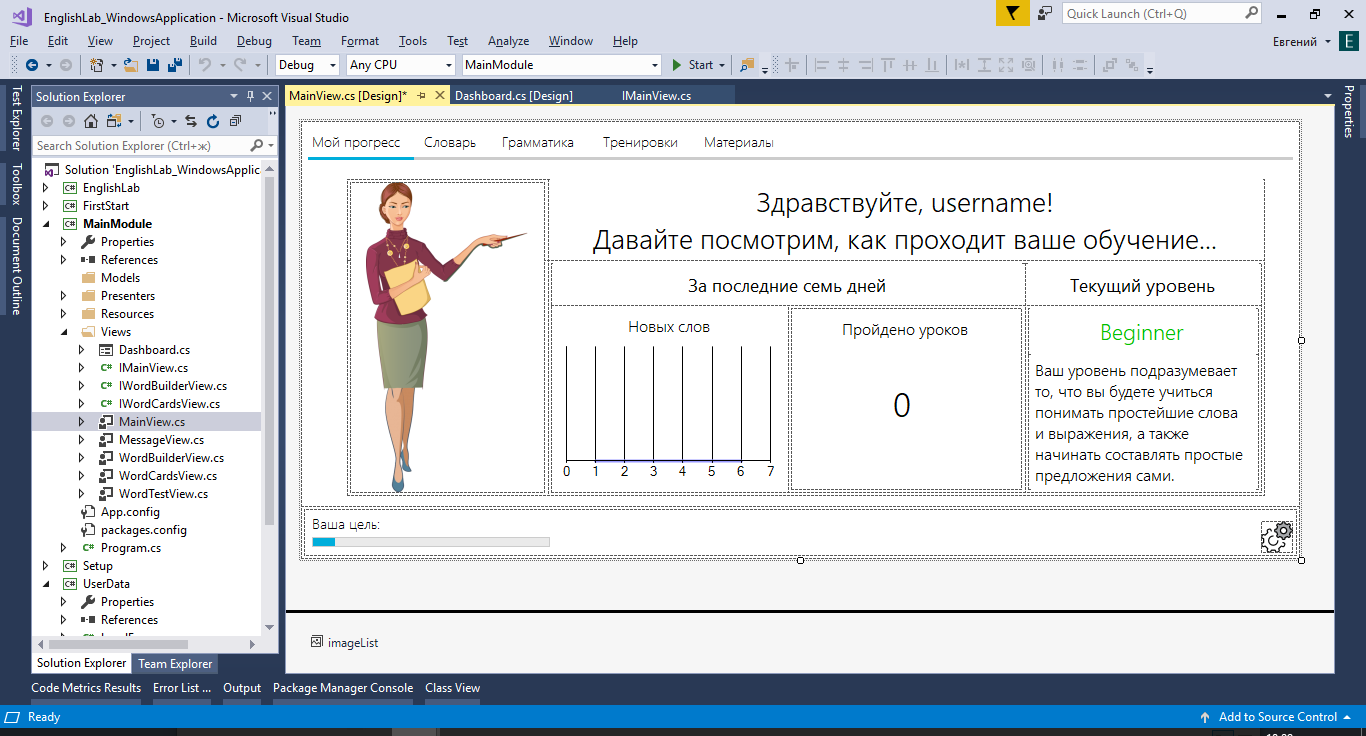


Рисунок 3.3.3 – Проектирование главного представления.

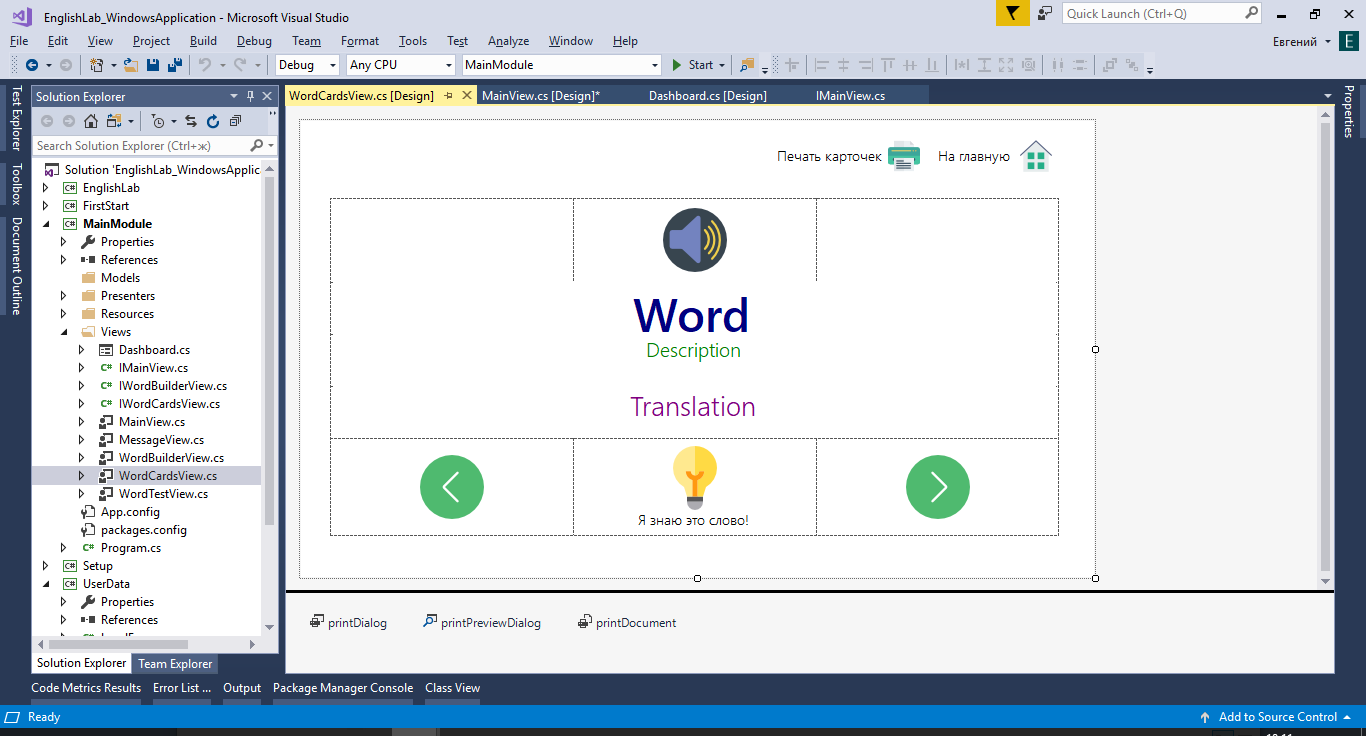


Рисунок 3.3.4 – Проектирование представления словарных карточек.

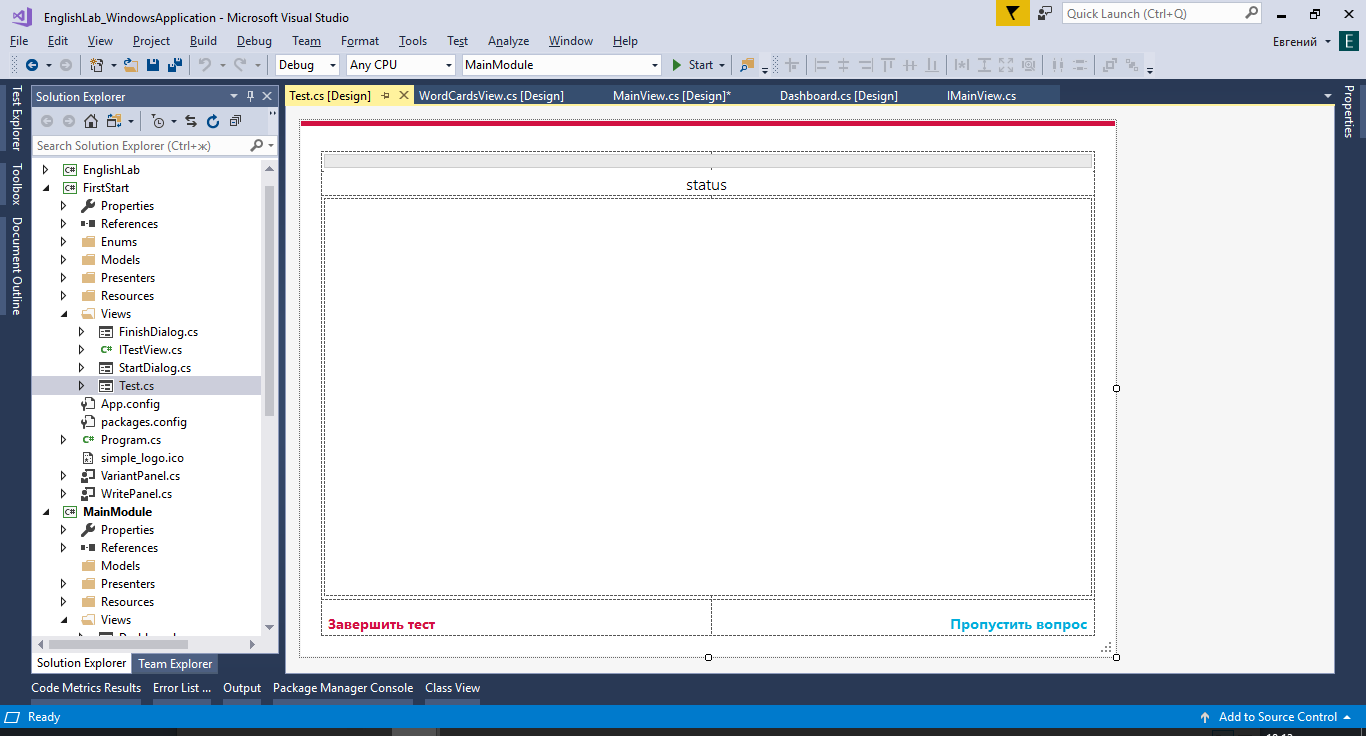


Рисунок 3.3.5 – Проектирование представления для тестирования.

После разработки представлений начинается разработка моделей и алгоритмов. На рисунке 3.3.6 изображена работа по разработке логики приложения (написание кода).

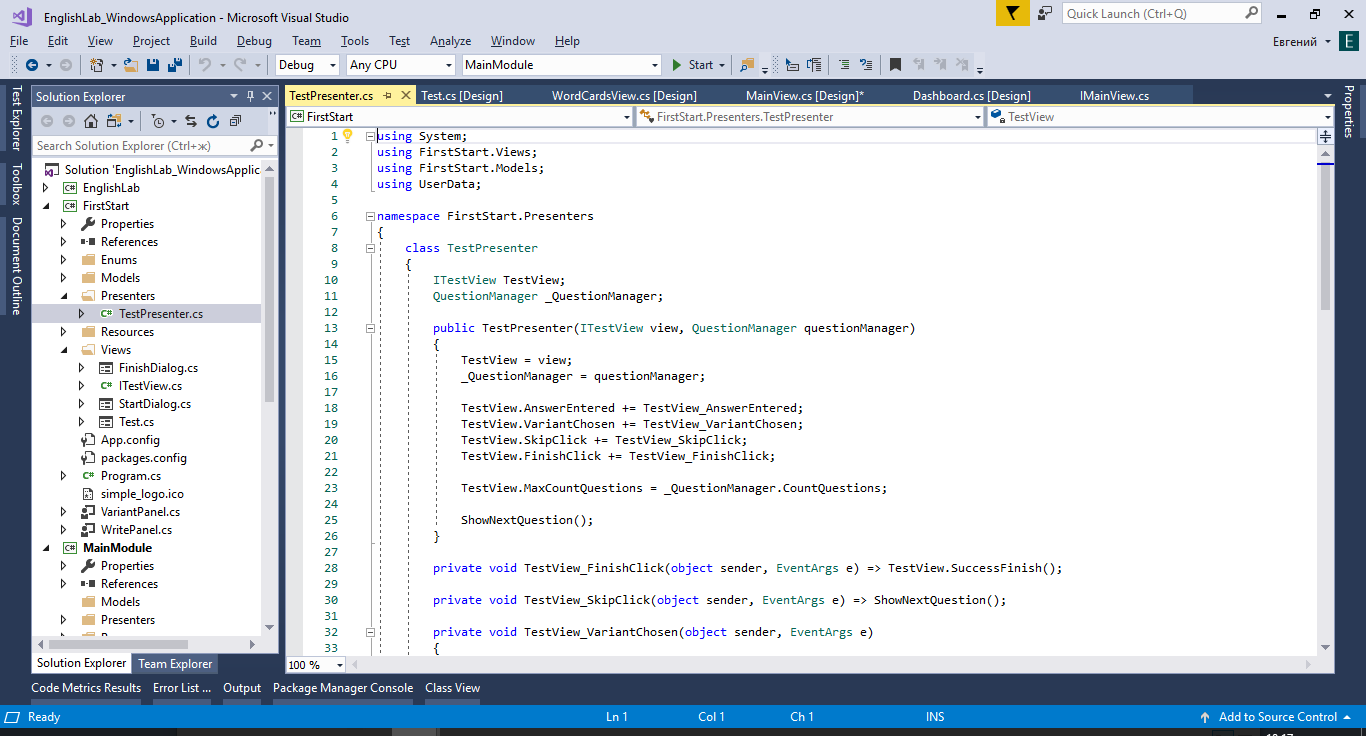


Рисунок 3.3.6 – Разработка логики приложения.

3.4 Паттерн проектирования

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

* проблема, которую решает паттерн;
* мотивации к решению проблемы способом, который предлагает паттерн;
* структуры классов, составляющих решение;
* примера на одном из языков программирования;
* особенностей реализации в различных контекстах;
* связей с другими паттернами.

Зачем нужно использовать паттерны:

* вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть открытием;
* вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены;
* вы произносите название паттерна, вместо того, чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.

Паттерны отличаются по уровню сложности, детализации и охвата проектируемой системы. Проводя аналогию со строительством, вы можете повысить безопасность перекрёстка, поставив светофор, а можете заменить перекрёсток целой автомобильной развязкой с подземными переходами.

Самые низкоуровневые и простые паттерны — идиомы. Они не универсальны, поскольку применимы только в рамках одного языка программирования.

Самые универсальные — архитектурные паттерны, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных её элементов.

Кроме того, паттерны отличаются и предназначением:

* порождающие паттерны беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей;
* структурные паттерны показывают различные способы построения связей между объектами;
* поведенческие паттерны заботятся об эффективной коммуникации между объектами.

При разработке данного проекта были задействованы два паттерна проектирования: паттерн Singleton и Model-View-Presenter (MVP).

Singleton — порождающий паттерн проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

На рисунке 3.4.1 представлена реализация паттерна Singleton.



Рисунок 3.4.1 – Реализация Singleton.

Использование паттерна Singleton при разработке данного приложения обусловлено тем, что логическое ядро приложения выделено в отдельный класс и это ядро является общим для всех модулей приложения. То есть, нам нет нужды каждый раз создавать экземпляр класса, содержащего логику, а достаточно просто создать его один раз и использовать везде, где он нам потребуется. Это позволяет сделать код более аккуратным, а также сэкономить память и вычислительные ресурсы.

Model-View-Presenter (MVP) — архитектурный паттерн проектирования пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения):

* модель (англ. Model) — хранит в себе всю логику, при необходимости получает данные из хранилища;
* вид (англ. View) — реализует отображение данных (из модели), обращается к Presenter за обновлениями;
* представитель (англ. Presenter) — реализует взаимодействие между моделью и представлением.

Обычно экземпляр Вида (Представления) создаёт экземпляр Представителя, передавая ему ссылку на себя. При этом Представитель работает с Видом в абстрактном виде, через его интерфейс. Когда вызывается событие Представления, оно вызывает конкретный метод Представителя, не имеющего ни параметров, ни возвращаемого значения. Представитель получает необходимые для работы метода данные о состоянии пользовательского интерфейса через интерфейс вида и через него же передаёт в вид данные из модели и другие результаты своей работы.

На рисунке 3.4.2 изображена схема взаимодействия частей приложения при использовании паттерна MVP.

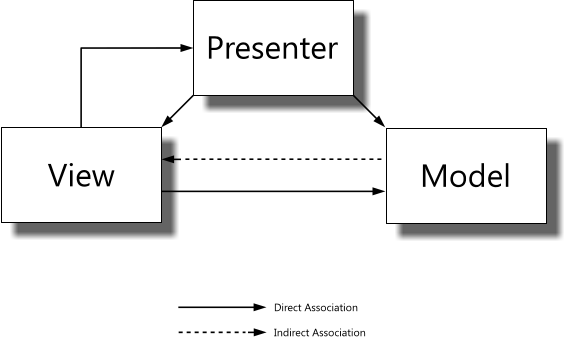


Рисунок 3.4.2 – Схема паттерна MVP.

Использование паттерна MVP при разработке данного приложения обусловлено тем, что приложение имеет большой объем кода. При разработке проектов, имеющих 2 и более форм, а также несущее в себе большой объем различной логики должно иметь различные части, которые являются самостоятельными единицами программы. Это значительно упрощает тестирование, модификацию и всю разработку приложения в целом, а также делает ваш код понятным всем программистам, знакомыми с MVP или MVC паттерном.

4 Модульное тестирование программного продукта и анализ кода

Модульное тестирование (Unit testing) – тестирование каждой атомарной функциональности приложения отдельно, в искусственно созданной среде. Именно потребность в создании искусственной рабочей среды для определенного модуля, требует от тестировщика знаний в автоматизации тестирования программного обеспечения, некоторых навыков программирования. Данная среда для некоторого юнита создается с помощью драйверов и заглушек.

Прежде всего, нужно очертить рамки, в которых Юнит-тестирование оправданно. Во-первых, архитектура проекта должна быть спроектирована в соответствии с идеями ООП (четкое деление на классы, каждый из которых выполняет свою определенную функцию), что обеспечит систему грамотным делением на модули. Также, модульное тестирование должно быть менее затратным при поиске дефектов, чем другие виды тестов и должно снижать время отладки кода.

Модульное тестирование мотивирует программистов писать код максимально оптимизированным, проводить рефакторинг (упрощение кода программы, не затрагивая ее функциональность), так как с помощью Юнит-тестирования можно легко проверить работоспособность рассматриваемого компонента.

Необходимость отделения реализации от интерфейса (ввиду особенностей модульного тестирования), что позволяет минимизировать зависимости в системе.

Документация Юнит-тестов может служить примером «живого документа» для каждого класса, тестируемого данным способом.

Модульное тестирование помогает лучше понять роль каждого класса на фоне всей программной системы.

Также, при «разработке через тестирование», которая активно используется в экстремальном программировании, модульное тестирования является одним из основных инструментов, позволяющий разрабатывать модули в соответствии с требованиями к данному модулю.

Пример прохождения модульного теста изображен на рисунке 4.1.

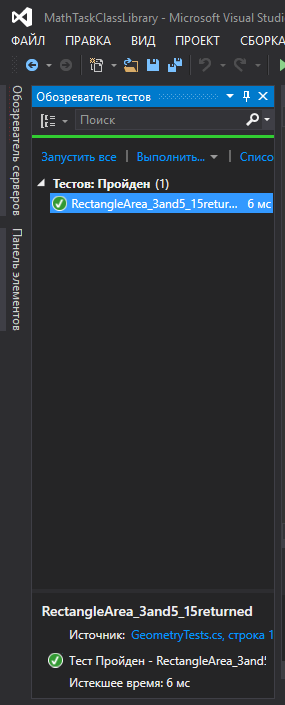


Рисунок 4.1 – Модульное тестирование.

При анализе кода решения получены метрики кода, изображенные на рисунке 4.2. Исходя их них можно сделать вывод о таких параметрах, как глубина наследования, зависимость классов друг от друга, сложность циклов, количество строк кода, а также убедиться в том, что код имеет высокий индекс удобства поддержки, о чем свидетельствуют оценки от 72 баллов до 90.

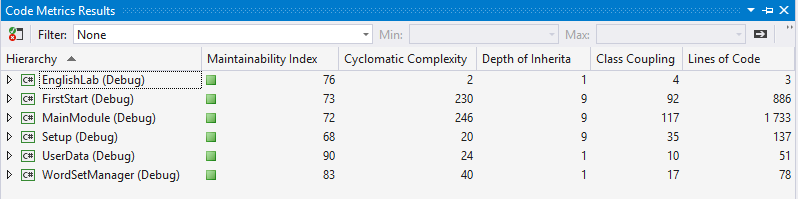


Рисунок 4.2 – Метрики кода.

5 Руководство пользователя

Запустив файл Setup.exe, вы увидите окно, изображенное на рисунке 5.1.

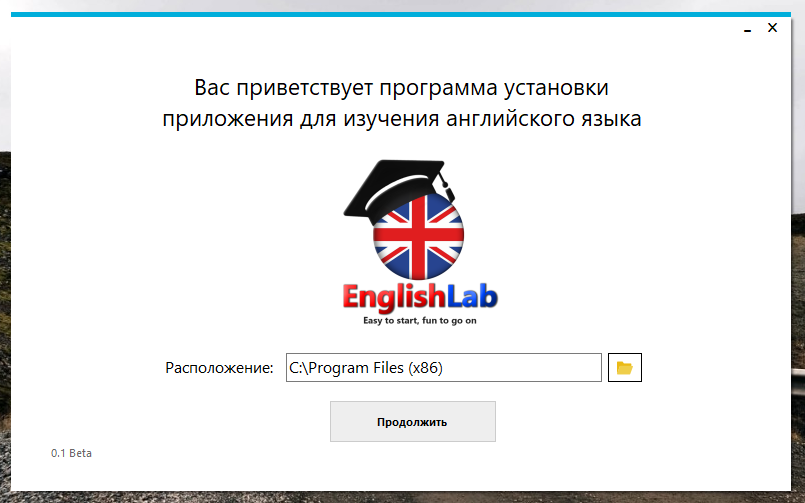


Рисунок 5.1 – Окно установки.

Вы можете выбрать папку для установки, нажав на значок папки (Рисунок 5.2).

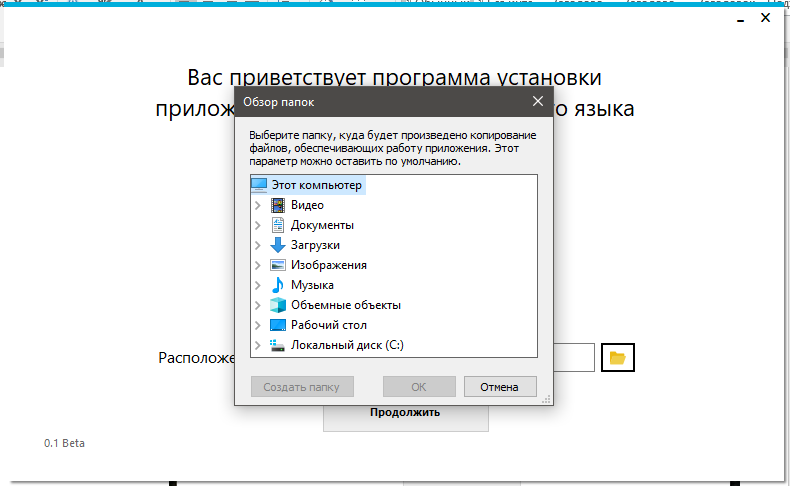


Рисунок 5.2 – Окно выбора папки.

По завершению установки запустите программу с рабочего стола, затем, следуя подсказкам на экране заполните информацию о себе.

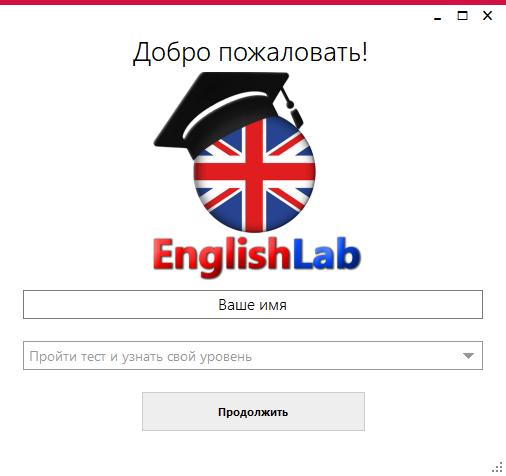


Рисунок 5.3 – Окно первого запуска.

Пройдите тест, если вы выбрали данную опцию (Рисунок 5.4).

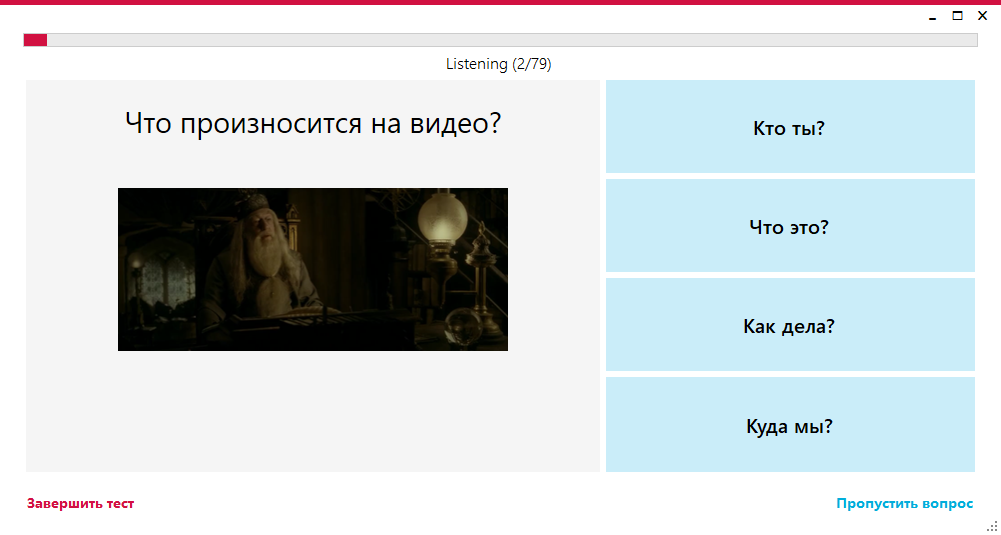


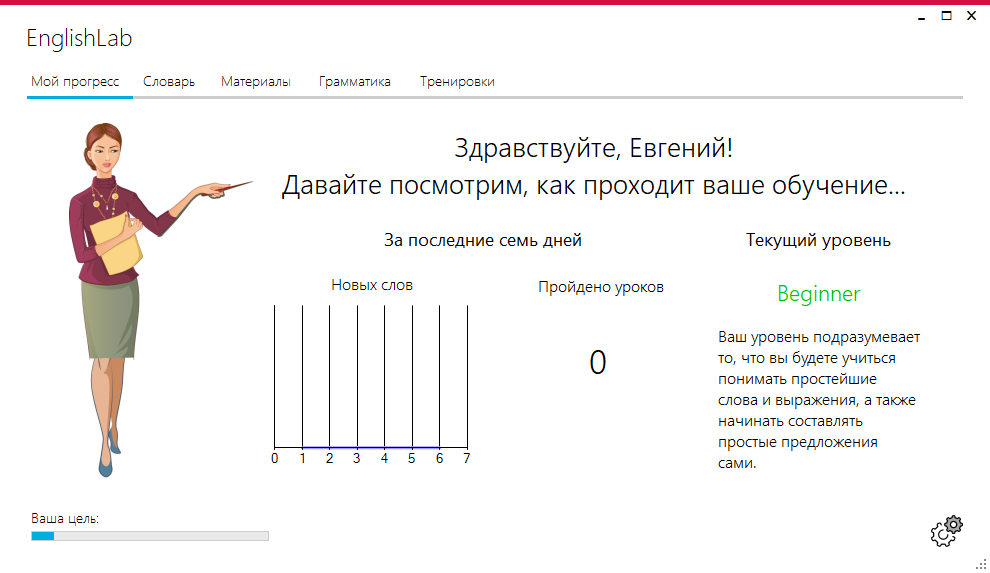
Рисунок 5.4 – Окно первого запуска.

На рисунке 5.5 представлено окно результатов тестирования.



Рисунок 5.5 – Результаты теста.

На рисунке 5.6 представлено главное окно приложения.

Рисунок 5.6 – Главное окно.

На главном окне представлена статистика. В верхней части располагаются вкладки, названные в соответствии с тем, что находится в представлениях, которые они открывают.

В окне «Словарь» (Рисунок 5.7) можно выбрать наборы слов, которые вы планируете изучить, а также отредактировать свой собственный набор. Выбранные для изучения наборы слов будут доступны в окне «Тренировки».

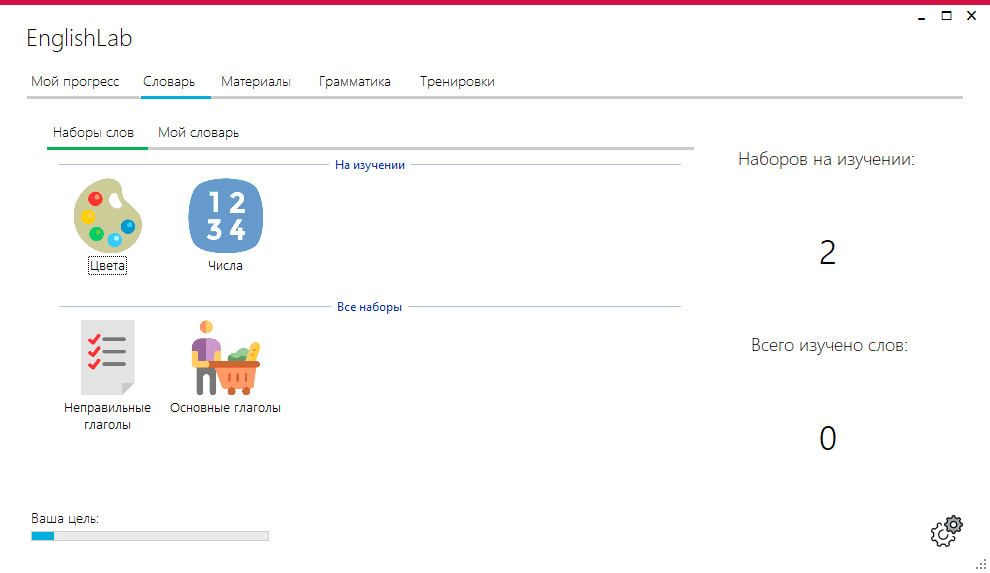
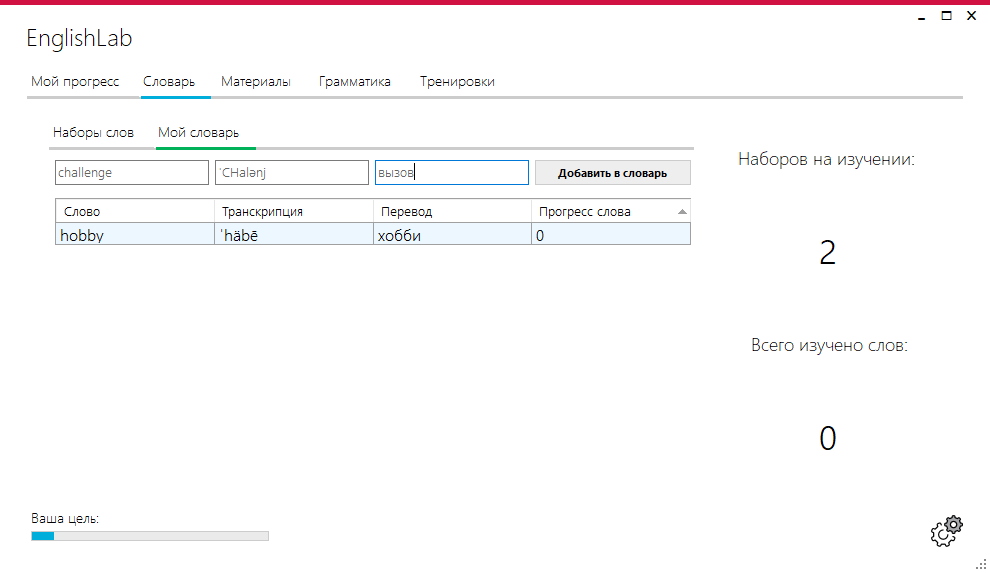
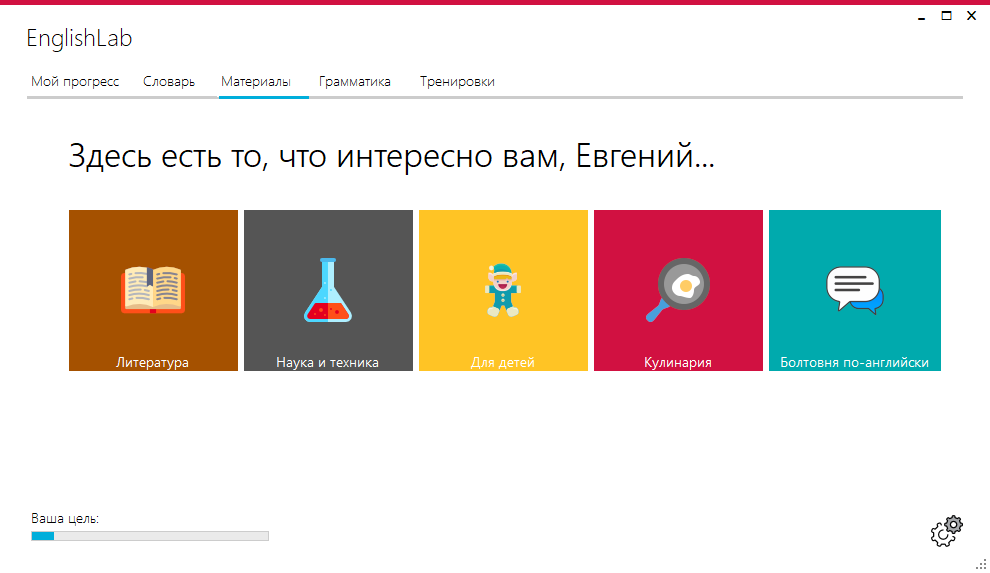


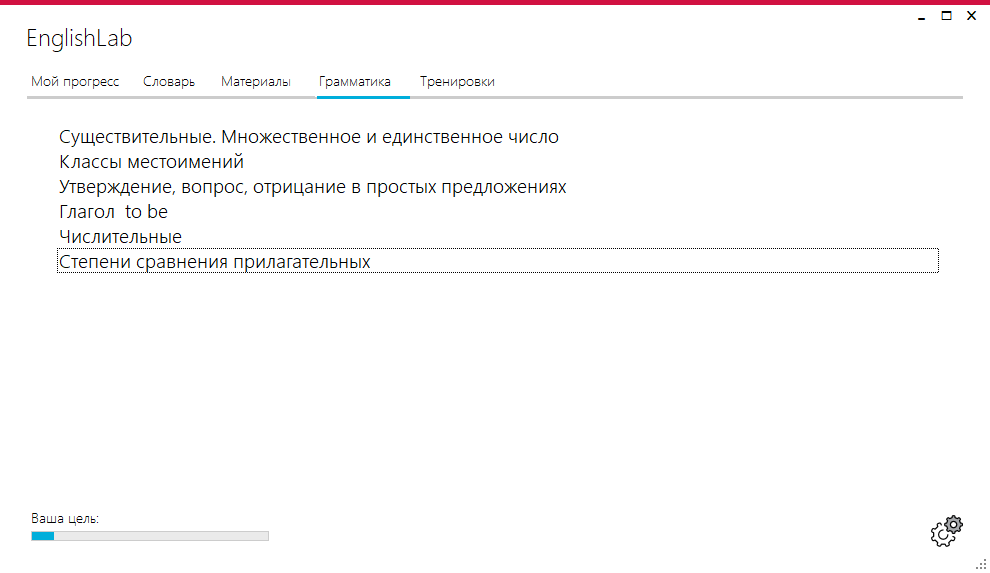
Рисунок 5.7 – Окно «Словарь».

Рисунок 5.8 – Окно «Редактирование пользовательского словаря».

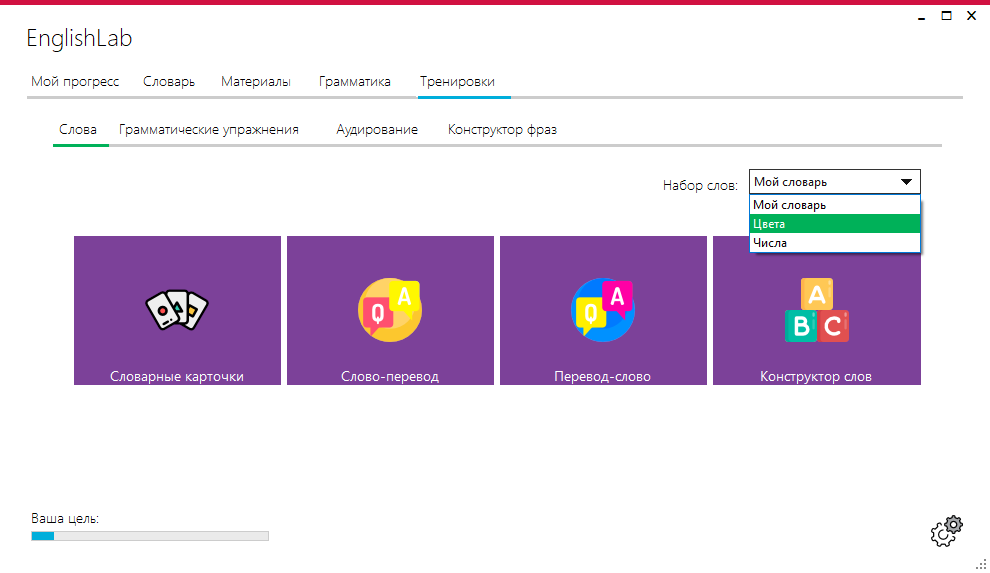
В окне «Материалы» (Рисунок 5.9) можно выбрать материал для чтения. Данный раздел на момент написания данной работы является не законченным и в дальнейшем будет постоянно пополняться.

Рисунок 5.9 – Окно «Материалы».

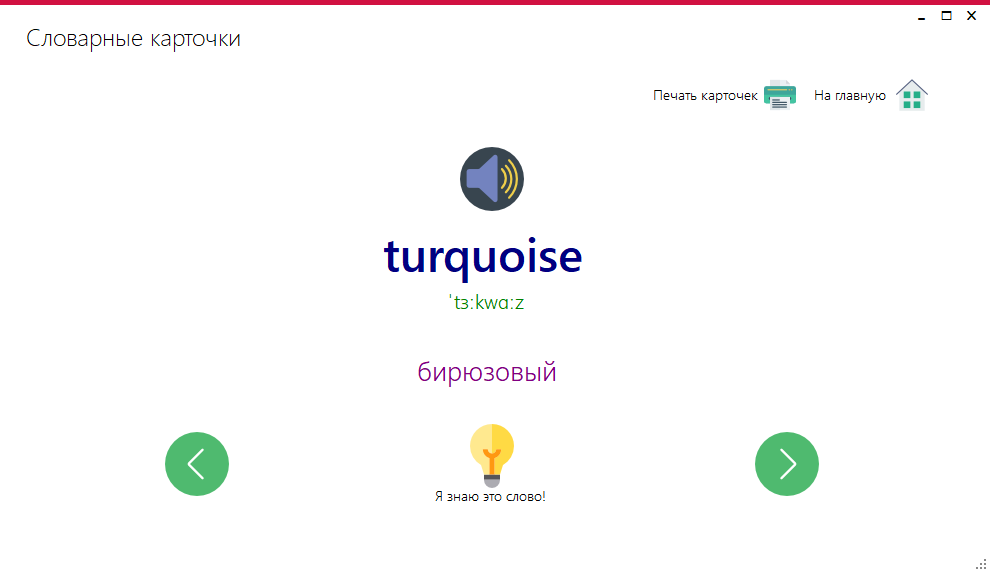
На рисунке 5.10 изображено окно «Грамматика», где предлагается выбрать тему для изучения.

  
Рисунок 5.10 – Окно «Грамматика».

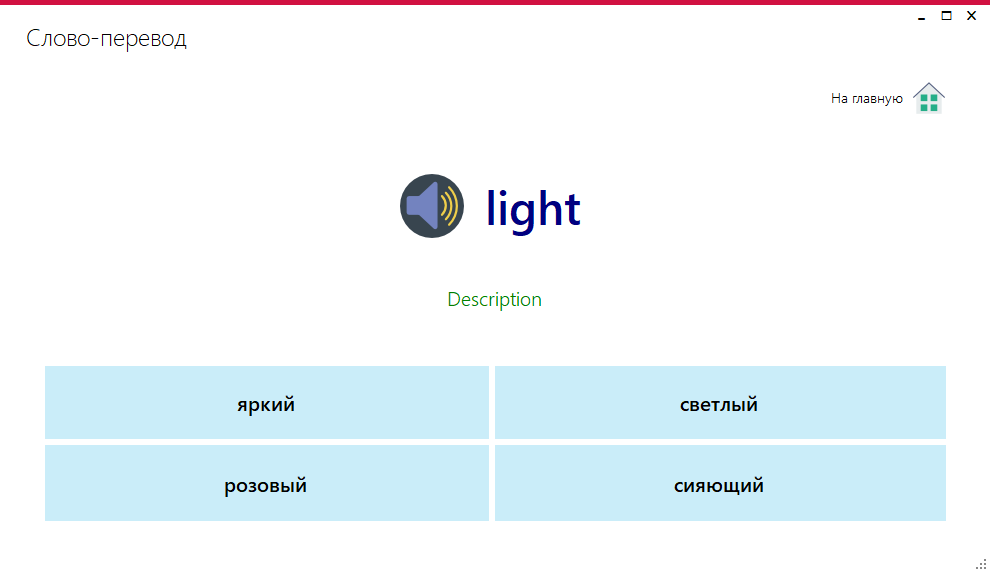
На рисунке 5.11 изображен раздел «Тренировки», в котором можно учить слова.

Рисунок 5.11 – Окно «Тренировки».

На рисунке 5.12 изображена работа тренировки словарных карточек.

Рисунок 5.12 – Окно «Словарные карточки».

На рисунке 5.13 изображена работа тренировки «Слово-перевод».

Рисунок 5.13 – Окно «Слово-перевод».

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был проведён анализ научной, методической литературы, интернет-порталов.

В результате дипломного проектирования создано приложение для изучения английского языка. Данное приложение предназначено для облегчения обучения школьников, студентов и всех желающих. Приложение разработано таким образом, чтобы пользователь смог закрепить или самостоятельно изучить предложенные темы, выучить новые слова, узнать свой уровень владения английским языком.

В ходе работы выполнены следующие задачи:

* изучена предметная область;
* разработано Windows приложение;
* разработано техническое задание;
* спроектированы диаграммы классов;
* применен объектно-ориентированный подход к программированию;
* приложение спроектировано в соответствии с Model-View-Presenter (MVP) и Singleton паттернами;
* код написан с высоким индексом удобства поддержки;
* разработан современный интерфейс пользователя;
* использованы модульные тесты для тестирования модели;
* разработан инсталлятор;

При необходимости приложение может быть дополнено или переработано, интерфейс и дизайн усовершенствованы.

Среди аналогов данное приложение выделяется простотой использования, малым размером и бесплатностью.

Таким образом, цель работы достигнута, поставленные задачи решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Зиборов В.В. Visual C# 2012 на примерах. – СПб. БХВ-Петербург, 2013. – 480 с.
2. Культин Н. Б. Visual C# в задачах и примерах. - СПб. БХВ-Петербург, 2015. – 320 с.
3. Пахомов Б. И. C# для начинающих. - СПб. БХВ-Петербург, 2014.   
   – 432 с.
4. Евдокимов П. В. C# на примерах. – СПб. Наука и Техника, 2016. – 304 с.
5. Спрол Антон. Думай, как программист: креативный подход к созданию кода. – Москва. Эксмо, 2018. – 272 с.
6. Русско-английский (британский) тематический словарь. 5000 слов. Международная транскрипция. – М. T&P Books Publishing, 2013. – 154 c.
7. Перри Грег. Программирование на C для начинающих. – Москва. Эксмо, 2015. – 368 с.
8. Васильев К. Б. Useful English. Полезный английский. – М. Астрель, 2013. – 382 с.
9. Гольцова Е. В. Английский язык для пользователей ПК и программистов. – СПб. Корона-век, 2016. – 512 с.
10. Фишман Л. М. Professional English. – М. Инфра-М, 2014. – 120 с.
11. Камянова Т. English grammar. – М. Дом славянской книги, 2015.   
    – 1024 с.
12. Гринченков Д. В. Математическая логика и теория алгоритмов для программистов. – М. Кнорус, 2013. – 206 с.
13. Угарова Е. В. Все времена английских глаголов. – М. Айрис-пресс, 2017. – 96 с.
14. Угарова Е. В. Все модальные глаголы английского языка. – М. Айрис-пресс, 2015. – 96 с.
15. Угарова Е. В. Английские предлоги. – М. Айрис-пресс, 2016. – 96 с.

ПРИЛОЖЕНИЕ А

Листинг управляющего кода

Test presenter:

using System;

using FirstStart.Views;

using FirstStart.Models;

using UserData;

namespace FirstStart.Presenters

{

class TestPresenter

{

ITestView TestView;

QuestionManager \_QuestionManager;

public TestPresenter(ITestView view, QuestionManager questionManager)

{

TestView = view;

\_QuestionManager = questionManager;

TestView.AnswerEntered += TestView\_AnswerEntered;

TestView.VariantChosen += TestView\_VariantChosen;

TestView.SkipClick += TestView\_SkipClick;

TestView.FinishClick += TestView\_FinishClick;

TestView.MaxCountQuestions = \_QuestionManager.CountQuestions;

ShowNextQuestion();

}

private void TestView\_FinishClick(object sender, EventArgs e) => TestView.SuccessFinish();

private void TestView\_SkipClick(object sender, EventArgs e) => ShowNextQuestion();

private void TestView\_VariantChosen(object sender, EventArgs e)

{

\_QuestionManager.CheckAnswer(TestView.SelectedVariant);

TestView.Status = \_QuestionManager.RightGrammar.ToString();

ShowNextQuestion();

}

private void TestView\_AnswerEntered(object sender, EventArgs e)

{

\_QuestionManager.CheckAnswer(TestView.EnteredAnswer);

TestView.Status = \_QuestionManager.RightGrammar.ToString();

ShowNextQuestion();

}

private void ShowNextQuestion()

{

try

{

Question question = \_QuestionManager.GetNextQuestion();

TestView.Mode = question.Mode;

TestView.Question = question.TextQuestion;

if (question.Mode == Mode.Choose)

TestView.Variants = question.Variants;

TestView.Progress++;

TestView.Status = String.Format("{0} ({1}/{2})", question.Skill.ToString(), TestView.Progress, \_QuestionManager.CountQuestions);

}

catch

{

TestView.SuccessFinish();

}

}

}

}

MainViewPresenter:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using UserData;

using WordSetManager;

using MainModule.Views;

namespace MainModule.Presenters

{

public class MainViewPresenter

{

IMainView MainView;

User ActiveUser;

public MainViewPresenter(IMainView mainView)

{

MainView = mainView;

ActiveUser = User.GetUser("UserInfo.dat");

//ActiveUser = new User("ff", Level.Elementary);

MainView.UserName = ActiveUser.ToString();

foreach (WordSet set in WordSet.GetList("WordSets"))

mainView.AddOrUpdateWordSet(set, set.Picture, "allGroup");

foreach (WordSet set in ActiveUser.SetsOnLearning)

{

if (set.ToString() == "Мой словарь") continue;

mainView.AddOrUpdateWordSet(set, set.Picture, "onLearningGroup");

}

MainView.AddWord += MainView\_AddWord;

MainView.TakeSetToLearn += MainView\_TakeSetToLearn;

MainView.RemoveSetAtLearning += MainView\_RemoveSetAtLearning;

UpdateInfo();

}

private void MainView\_RemoveSetAtLearning(object sender, EventArgs e)

{

ActiveUser.RemoveSetFromLearning(MainView.SelectedWordSet);

MainView.UpdateWordSet(MainView.SelectedWordSet, "allGroup");

UpdateInfo();

}

private void MainView\_TakeSetToLearn(object sender, EventArgs e)

{

ActiveUser.AddSetToLearn(MainView.SelectedWordSet);

MainView.UpdateWordSet(MainView.SelectedWordSet, "onLearningGroup");

UpdateInfo();

}

private void MainView\_AddWord(object sender, EventArgs e)

{

ActiveUser.AddWordInUserDictiornary(MainView.Word, MainView.Translation, MainView.Transcription);

UpdateInfo();

}

private void UpdateInfo()

{

MainView.CountLearntWords = ActiveUser.CountLearntWords;

MainView.UserVocabulary = ActiveUser.UserDictornary.GetDataTable();

MainView.CountWordSetsOnLearning = ActiveUser.CountSetsOnLearning;

MainView.WordSetsOnLearning = ActiveUser.SetsOnLearning;

ActiveUser.Save();

}

}

}

WordCardsPresenter:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using MainModule.Views;

using WordSetManager;

namespace MainModule.Presenters

{

class WordCardsPresenter

{

private IWordCardsView CardsView;

private WordSet WordSet;

private Word CurrentWord;

public WordCardsPresenter(IWordCardsView cardsView, object wordSet)

{

CardsView = cardsView;

WordSet = wordSet as WordSet;

CardsView.NextClick += CardsView\_NextClick;

}

private void CardsView\_NextClick(object sender, EventArgs e)

{

Word word = WordSet.GetNextWord();

CardsView.Word = word.ToString();

CardsView.Translation = word.Translation;

CardsView.Description = word.Description;

}

}

}

ПРИЛОЖЕНИЕ Б

Окна

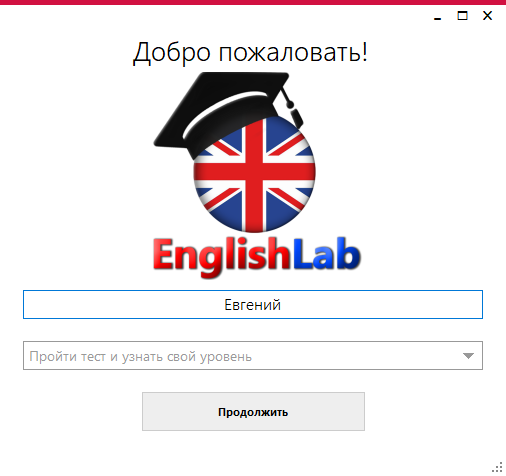


Рисунок 1 – Окно первого запуска.

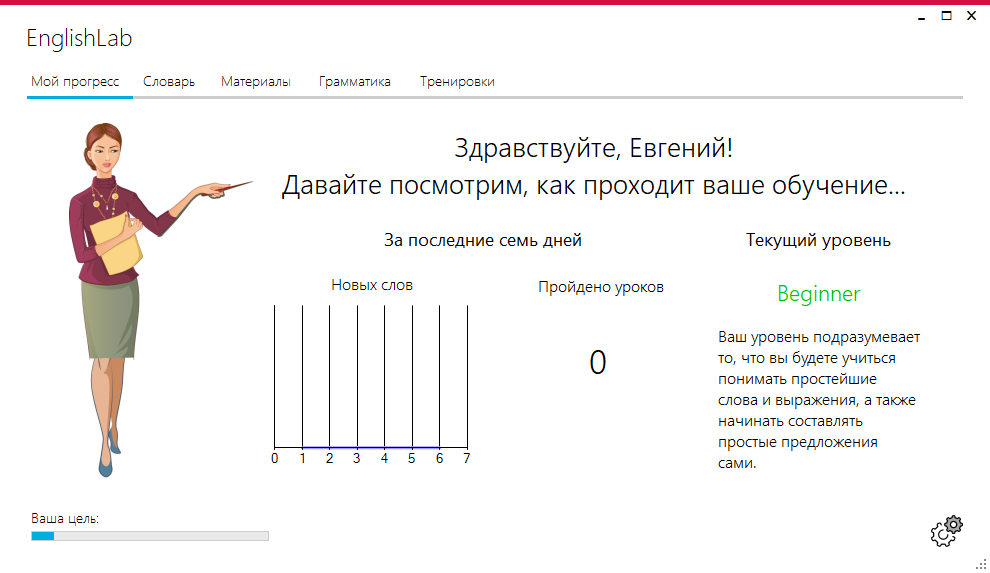


Рисунок 2 – Главное окно.

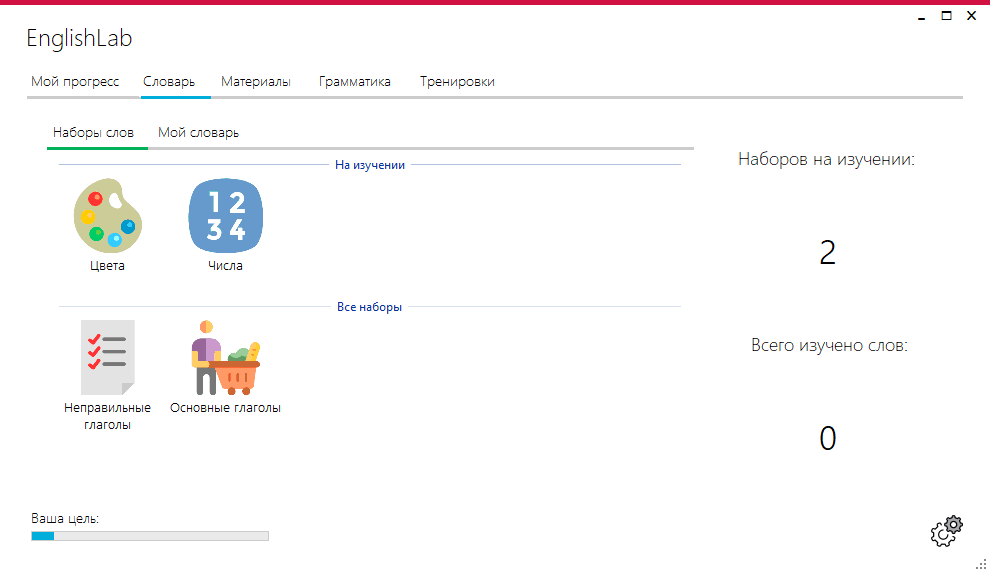


Рисунок 3 – Окно словаря.

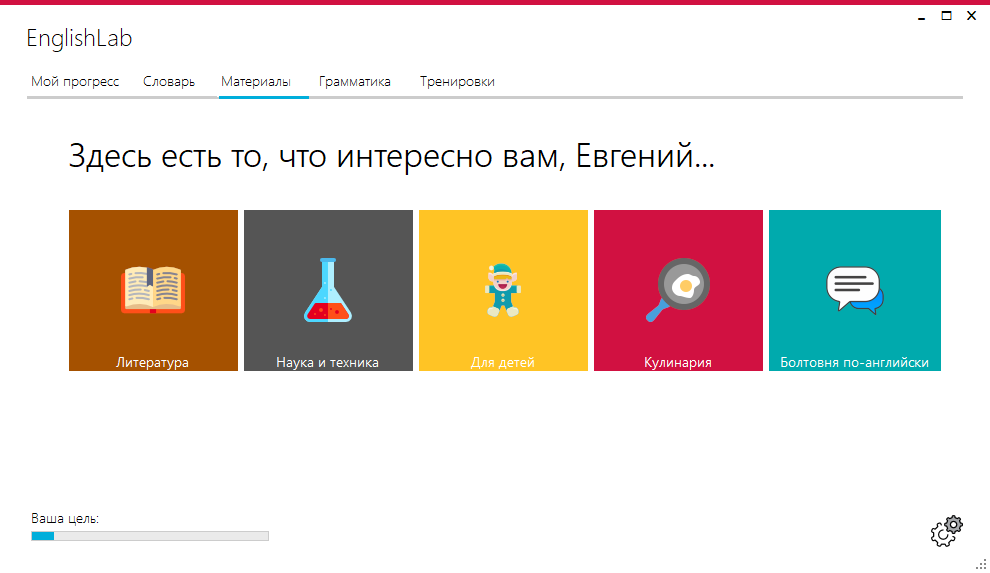


Рисунок 4 – Окно материалов.

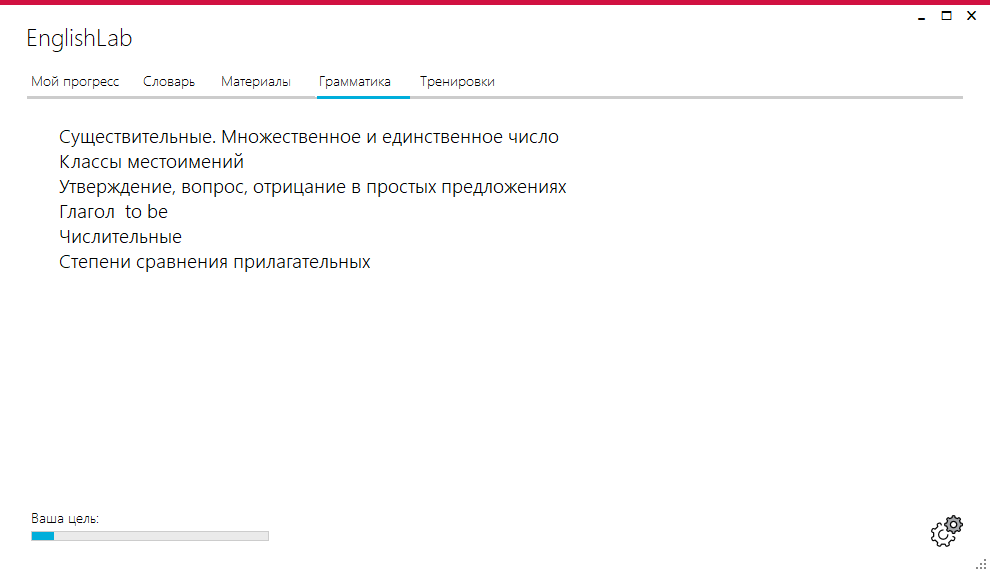


Рисунок 5 – Окно грамматики.

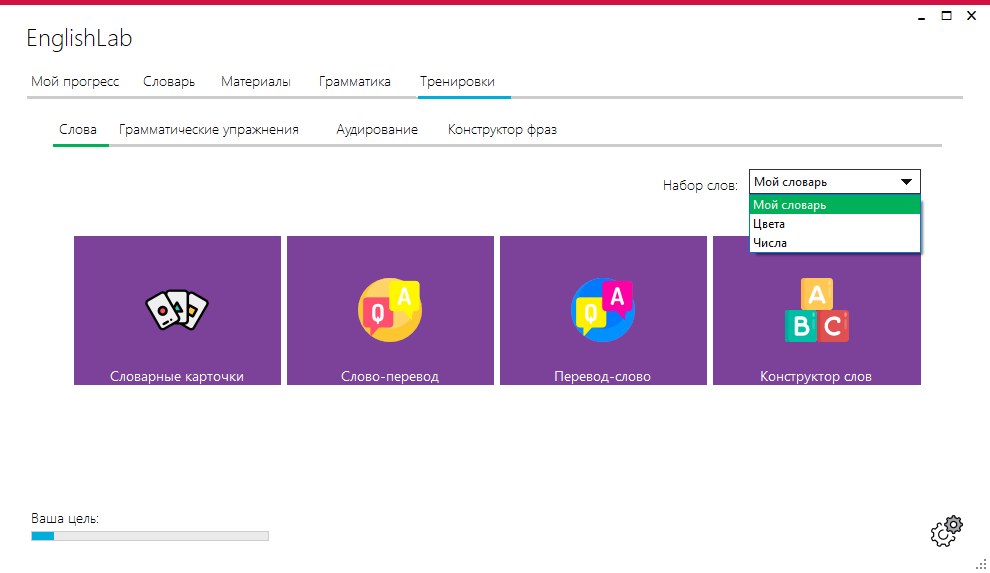


Рисунок 6 – Окно тренировок.



Рисунок 7 – Окно карточек.