1. Разработка программных модулей в среде MS Visual Studio 2019

1.1. Изучение постановки задачи(Назначение разработки)

Программный продукт «Victorina» – программа, созданная для работы с базой данных по предметной области.

Автоматизация продукта «Victorina» - процесс внедрения программно-аппаратных комплексов для облегчения и автоматизации.

Приложение должно выполнять следующие функции:

• Вход в программу при запуске

• Просмотр информации из базы данных

• Обеспечить идентицифированным пользователям функционал по их ролям

• Корректно выводить данные из базы данных.

• Добавление новой информации

• Удаление информации

• Изменение информации

• Сортировка

• Переходы по страницам

1.2. Описание области применения и пользователей приложения (use-case диаграмма с ее описанием)

**Пользователи:**

1. **Студент**

Роль «Студент» в приложении имеет доступ ко следующим функциям: «Прохождение тестов», «Просмотр своих результатов», «Управление личной информацией»

1. **Админ**

Роль «Админ» в приложении имеет доступ ко следующим функциям: «Просмотр чужих результатов», «Удаление/редактирование/добавление новых пользователе».

Рис №1. Сase-диаграмма

1.3. Проектирование интерфейса приложения

1.3.1. Правила и принципы разработки интерфейса (теория)

**Главная задача — ясность**

Ясность — это первая и самая главная задача любого интерфейса. Чтобы интерфейс эффективно помогал людям добиваться своих целей, он должен обладать следующими характеристиками.

* Во-первых, он должен быть узнаваемым, а его назначение — очевидным для пользователя.
* Во-вторых, люди должны понимать, с чем они взаимодействуют через интерфейс.
* Наконец, процесс взаимодействия с интерфейсом должен быть предсказуемым. В интерфейс можно внести некую загадочность или элементы игры, но вот путаницы быть не должно.

Ясность рождает в пользователях уверенность и готовность продолжать работу с интерфейсом. Сто понятных экранов лучше, чем один беспорядочный.

# Ключевое назначение интерфейсов — взаимодействие

Интерфейсы существуют, чтобы люди могли взаимодействовать с нашим миром. Через интерфейс мы можем прояснить, проиллюстрировать, дать возможность, показать взаимосвязь, объединить людей или разделить, управлять ожиданиями и давать доступ к услугам. Процесс проектирования интерфейсов — это не Искусство.

*Интерфейсы — это не памятники самим себе. Интерфейсы выполняют конкретные задачи, их эффективность измерима.*

Однако, они могут выходить за рамки чисто практического применения. Лучшие интерфейсы — те, которые вдохновляют, пробуждают чувства, удивляют и усиливают наш опыт общения с миром.

# Удержать внимание любой ценой

Мы живем в мире отвлекающих факторов. В наше время даже почитать спокойно не получится — кто-то или что-то всегда будет отвлекать нас, перетягивать наше внимание. Внимание — это большая ценность.

1.3.2. Разработка макета приложения

**Страница авторизации.**

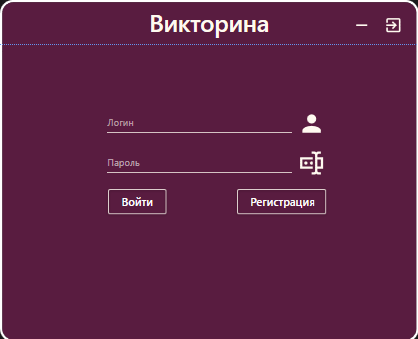


Рис №2 Страница авторизации

**Страница Пользователя.**



Рис №3 Страница пользовательского интефейса

**Страница Админа.**



Рис №4 Страница админского интефейса

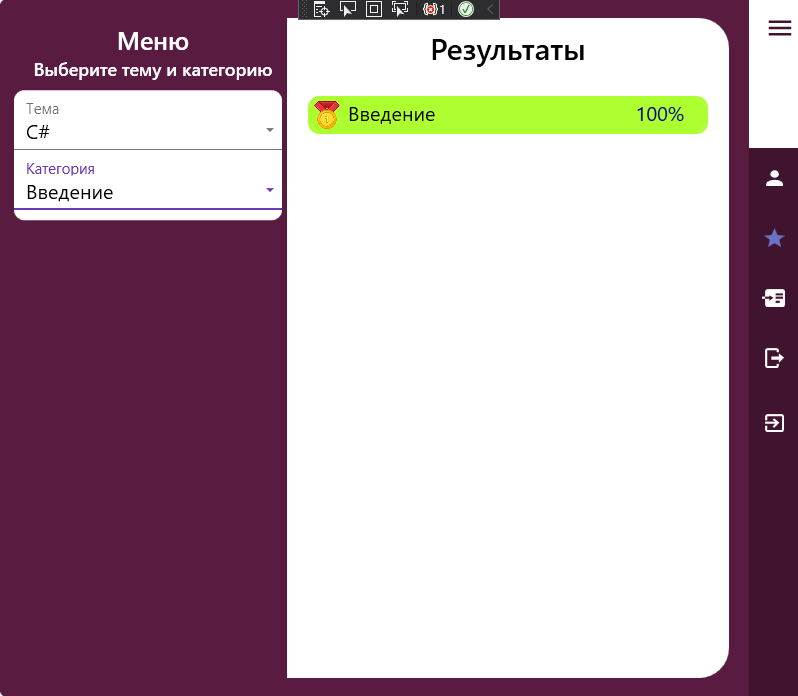
**Страница Результатов.** 

Рис №5 Страница результатов(Пользователь)

**Страница профиля.**

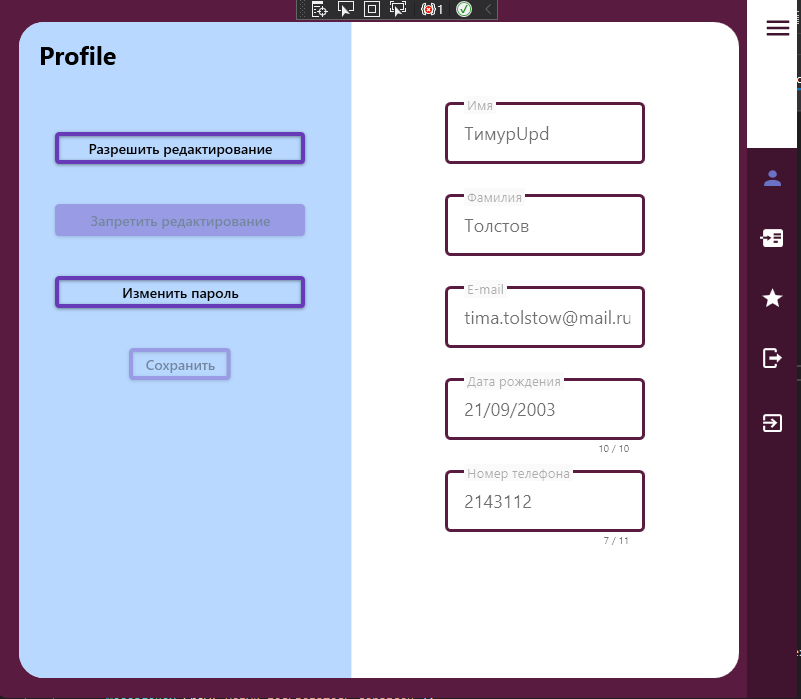


Рис №6 Страница профиля

**Окно регистрации.**

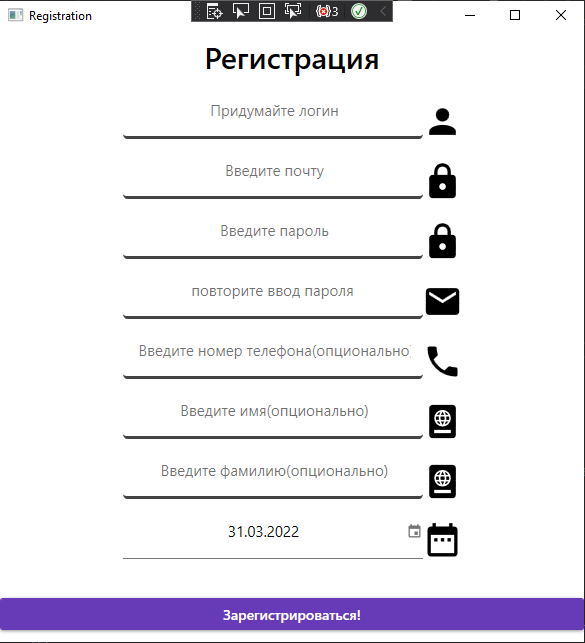


Рис №7 Страница регистрации

**Окно Изменения пользователя(Админ)**

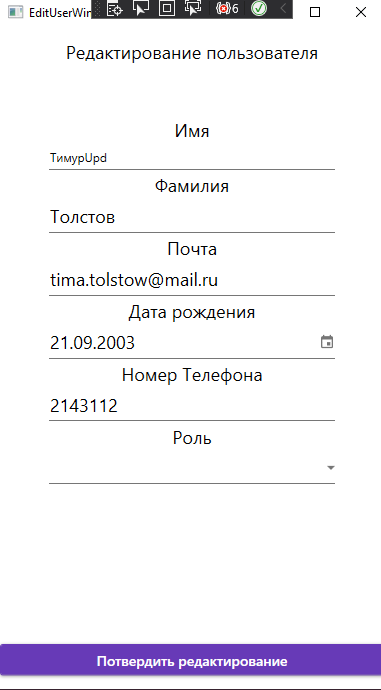


Рис №8 окно редактирование пользователя

**Окно добаления пользователя(Админ)**

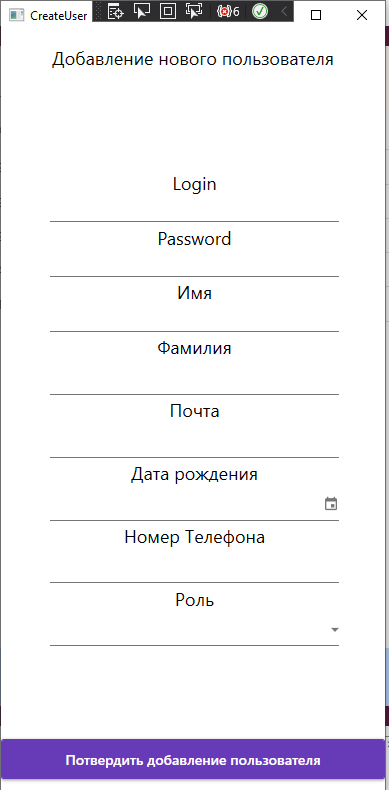


Рис №9 окно добавления пользователя

**Страница теста.**

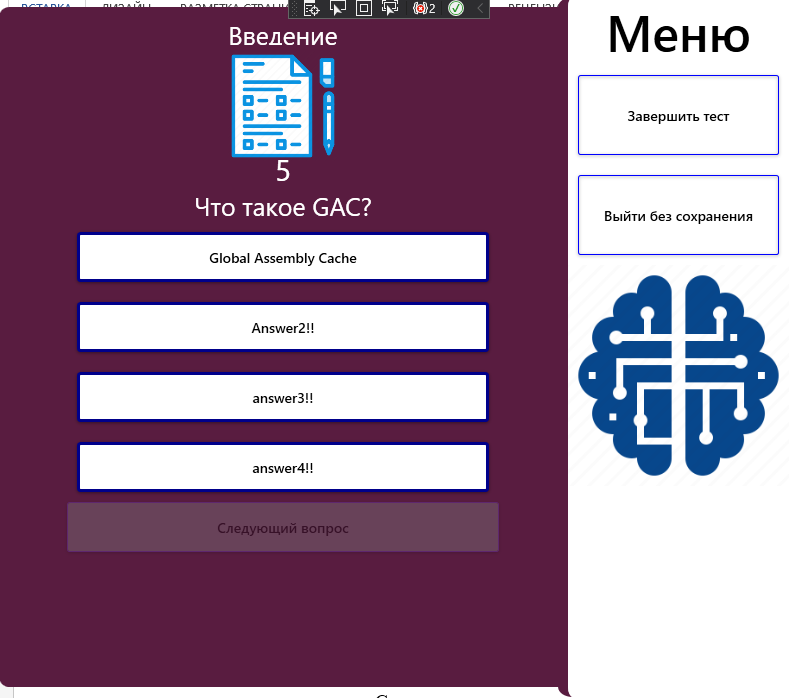


Рис №10 Страница прохождения теста

1.4. Стандарты оформления кода языка C#

1.4.1. Соглашения об именовании (теория по camel-case)

Camel case (иногда стилизованный под camelCase или camelCase, также известный как camel caps или более формально как медиальные столицы) - это практика написания фраз без пробелов или знаков препинания, указывающая на разделение слов с одной заглавной буквой и первое слово, начинающееся с любого случая. Распространенные примеры включают "iPhone" и "eBay". Он также иногда используется в онлайн-именах пользователей, таких как "JohnSmith", и для того, чтобы сделать многословные доменные имена более разборчивыми, например, при продвижении "EasyWidgetCompany.com.

Camel case часто используется в качестве соглашения об именах в компьютерном программировании, но является неоднозначным определением из-за необязательной капитализации первой буквы. Некоторые стили программирования предпочитают camel case с заглавной буквой, другие нет. Для ясности в этой статье называются две альтернативы верхний регистр верблюда (начальная заглавная буква, также известная как Начальные заглавные буквы, Начальные заглавные буквы, InitCaps или Pascal case) и нижний регистр верблюда (начальная строчная буква, также известная как падеж верблюда). Некоторые люди и организации, в частности Microsoft, используют этот термин camel case только для нижнего camel case, обозначая Pascal case для верхнего camel case.

1.4.2. Таблица идентификаторов программных объектов

|  |  |  |
| --- | --- | --- |
| Название обьекта | Тип обьекта | Назначение |
| Pages | | |
| AdminInterface | Страница | Интерфейс админа |
| AuthorizationPage | Страница | страница для авторизации |
| CategoryWnd | Окно | Окно категории |
| CreateNewUserWnd | Страница | Окно создание нового пользователя(админ) |
| EditUserWnd | Окно | Окно редактирования пользователя |
| ProfileUser | Страница | Страница профиля пользователя |
| RegistrationWnd | Окно | Окно регистрации |
| ResultPage | Страница | Страница результатов |
| TestForm | Страница | Страница прохождения тестов |
| UserInterface | Страница | Страница пользовательского интефейса |
| Services | | |
| СategoryServices | Класс | Запросы для коллекци категории |
| QuestionsServices | Класс | Запросы для коллекции вопросов |
| ResultsServices | Класс | Запросы для коллекции резульататов |
| TemaServices | Класс | Запросы для коллекции тем |
| TestItemServices | Класс | Запросы для коллекии тестов |
| UserService | Класс | Запросы для коллекции юзеров |

1.5. Создание и использование стилей в технологии WPF

1.5.1. Понятие и описание стиля

Стили позволяют определить набор некоторых свойств и их значений, которые потом могут применяться к элементам в xaml. Стили хранятся в ресурсах и отделяют значения свойств элементов от пользовательского интерфейса. Также стили могут задавать некоторые аспекты поведения элементов с помощью триггеров. Аналогом стилей могут служить каскадные таблицы стилей (CSS), которые применяются в коде html на веб-страницах.

1.5.2. Разработка стилей для элементов управления

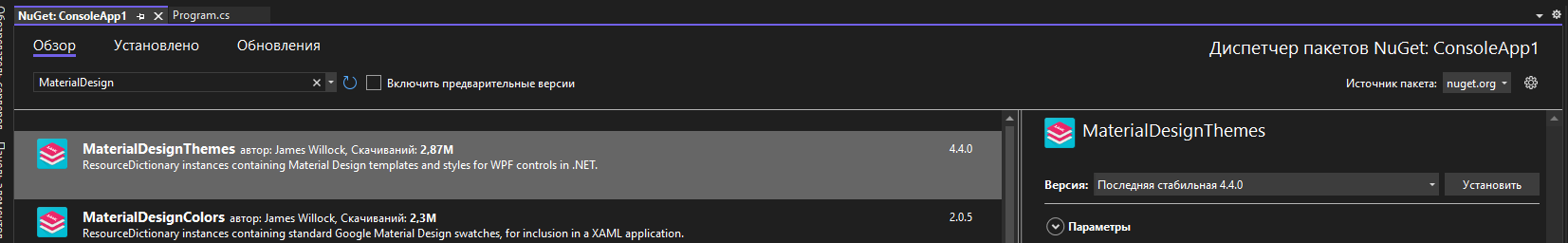


1.5.3. Применение стилей для оформления интерфейса приложения

Стили применяются автоматически при создании обьектов в коде xaml. Вид и цвет обьектов задается в app.xaml для всех обьектов используя nu-get пакет MaterilDesign.

Подключение библиотеки MaterialdDesign

* 1. пкм, по названию проекта в обозревателе решений
  2. «Управление Nu-get пакетами»
  3. В поиске вводим MaterialDesign и справа нажимаем установить



* 1. Установка завершена

1.5.4. Использование страниц Page в приложении

**Папка pages:**

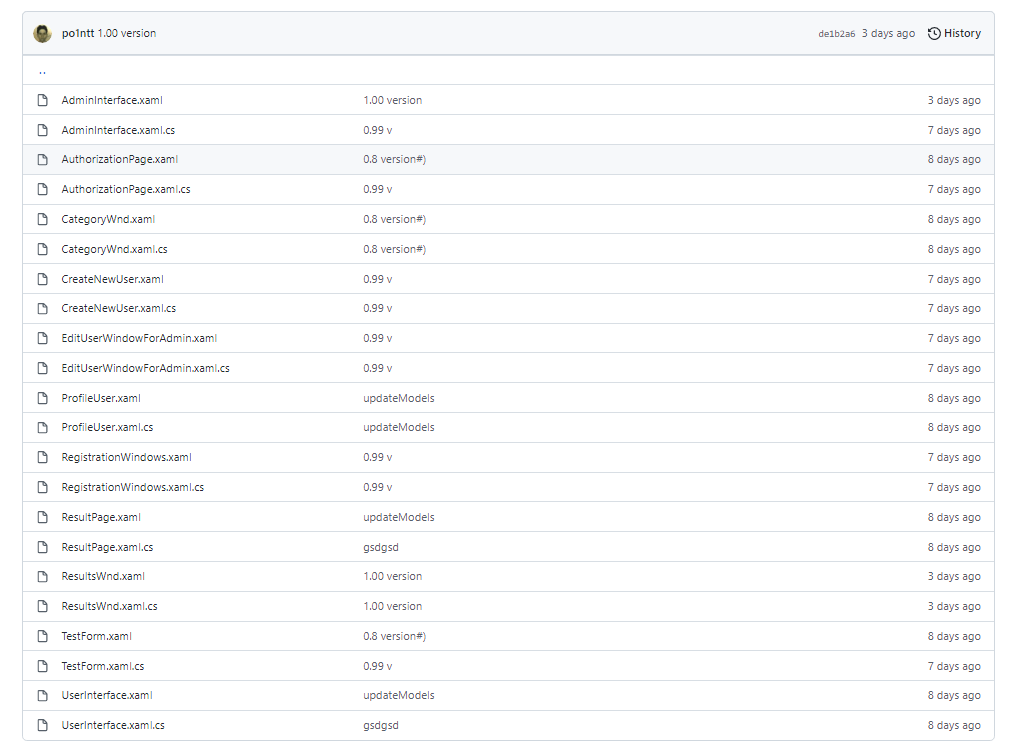


Рис №11. Папка Pages(В проекте)

**Класс навигации:**

class Frame

{

public static System.Windows.Controls.Frame FrameOBJ { get; set; }

}

**MainPage.cs:**

public MainWindow()

{

InitializeComponent();

ObjClas.Frame.FrameOBJ = MainFrame;

ObjClas.Frame.FrameOBJ.Navigate(new AuthorizationPage());

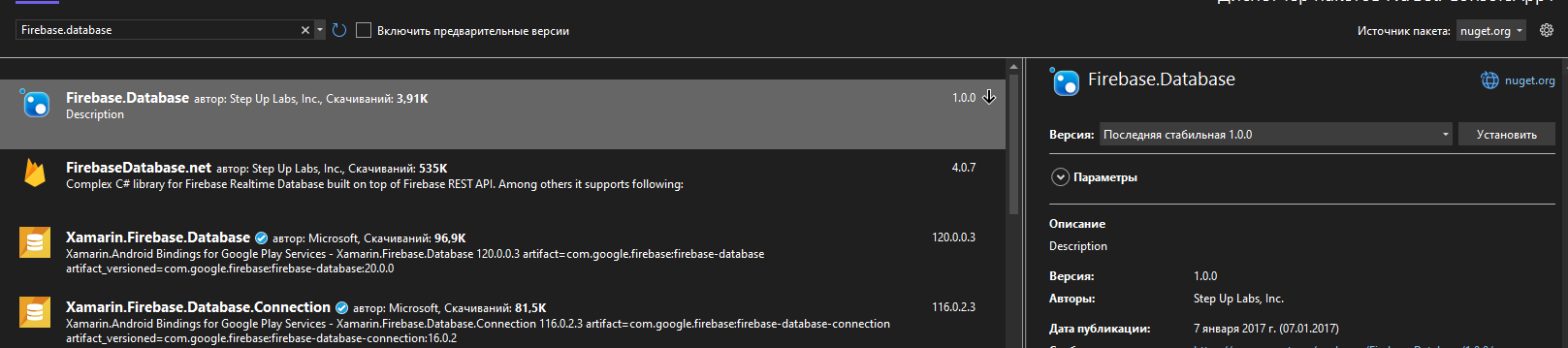
}

1.6. Организация доступа к данным из приложения

1.6.1. Использование Nu-get библиотеки FireBase.Database для связи с базой данных(теория)

Для установки этой библиотеки нужно проделать следующие действия:

* 1. пкм, по названию проекта в обозревателе решений
  2. «Управление Nu-get пакетами»
  3. В поиске вводим firebase.database и справа нажимаем установить



* 1. Установка завершена.

Как обращаться к базе данных через эту библиотеку описано ниже.

1.6.2. Создание приложения с применением Entity Framework (подключение к БД, модули отображения данных, добавления, редактирования, поиска, фильтрации)

Разработка базы данных проходила в веб-сервисе google. Firebase.Console.

База данных имеет следующие коллекции:

* **Сategories:**

1. Cathegoryid;
2. СategoryName
3. TemaId
4. ImageUrl

* **ItemLearnCategory**

1. ImageUrlItem
2. NameItemLearn
3. Url\_toLearningSite
4. Id\_LearnCategory
5. Id\_ItemLearn

* **LearningCategory**

1. ImageLearnCat
2. NameLearnCategory
3. TemaId
4. IdLearnCat

* **Questions**

1. TextQuest
2. Id\_quest
3. Id\_test
4. Quest\_Answer1
5. Quest\_answer2
6. Quest\_answer3
7. QuestAnaswer4
8. Quest\_rightanswer

* **Results**

1. CategoryId
2. MedalImage
3. NameTestDone
4. ScorePercennt
5. TestID
6. User\_Login

* **Role**

1. Id\_role
2. NameRole

* **Tema**

1. NameTema
2. TemaId

* **Tests**

1. CategoryId
2. Description
3. Name
4. RightAnswerId
5. TestId

* **Users**

1. BirthDay
2. Email
3. Login
4. Name
5. Phone
6. Password
7. RoleId
8. SurName

В веб-сервисе база данных выглядит следующим образом:



Подключения в базе данных осуществляется следующим программным кодом:

Using Firebase.Database;

FirebaseClient client;

public СathegoryServices()

{

client = new FirebaseClient("https://victorinaproject-default-rtdb.firebaseio.com/");

}

Примеры программное реализации функции Добавления/Редактирование/Удаления/Просмотр на примере одной из модели данных(Questions)

**1. Удаление:**

public async Task<bool> DeleteQuest(int Id\_Quest)

{

var keytodelete = (await client.Child("Questions").OnceAsync<Questions>()).FirstOrDefault(a => a.Object.id\_quest == Id\_Quest);

await client.Child("Questions").Child(keytodelete.Key).DeleteAsync();

return true;

}

**2. Добавление**

public async Task<bool> AddQuest(string textquest, string questanswer1, string questanswer2, string questanswer3, string questanswer4, string questrightanswer, TestsModel tests)

{

var questions = await GetQuestionsAsync();

await client.Child("Questions").PostAsync(new Questions()

{

TextQuest = textquest,

id\_quest = questions.Count + 1,

quest\_answer1 = questanswer1,

quest\_answer2 = questanswer2,

quest\_answer3 = questanswer3,

quest\_answer4 = questanswer4,

quest\_rightanswer = questrightanswer,

id\_test = tests.TestId

});

return true;

}

**3. Редактирование**

public async Task<bool> UpdateQuestions(int id\_Quest,string textquest, string questanswer1, string questanswer2, string questanswer3, string questanswer4, string questrightanswer, TestsModel tests)

{

var keyquest = (await client.Child("Questions")

.OnceAsync<Questions>())

.FirstOrDefault

(a => a.Object.id\_quest == id\_Quest);

Questions quest = new Questions() {TextQuest = textquest, id\_quest = id\_Quest, id\_test = tests.TestId, quest\_answer1 = questanswer1, quest\_answer2 = questanswer2, quest\_answer3 = questanswer3, quest\_answer4 = questanswer4, quest\_rightanswer = questrightanswer };

await client.Child("Questions")

.Child(keyquest.Key)

.PutAsync(quest);

return true;

}

**4. Просмотр**

public async Task<List<Questions>> GetQuestionsAsync()

{

var questions = (await client.Child("Questions").OnceAsync<Questions>()).Select(f => new Questions

{

id\_quest= f.Object.id\_quest,

id\_test = f.Object.id\_test,

quest\_answer1 = f.Object.quest\_answer1,

quest\_answer2 = f.Object.quest\_answer2,

quest\_answer3 = f.Object.quest\_answer3,

quest\_answer4 = f.Object.quest\_answer4,

TextQuest = f.Object.TextQuest,

quest\_rightanswer =f.Object.quest\_rightanswer

}).ToList();

return questions;

}

public async Task<ObservableCollection<Questions>> GetQuestionsAsyncBYTest(int test\_id)

{

var QuestionsByTest = new ObservableCollection<Questions>();

var items = (await GetQuestionsAsync()).Where(p => p.id\_test == test\_id).ToList();

foreach (var item in items)

{

QuestionsByTest.Add(item);

}

return QuestionsByTest;

}

Работа с другими коллекциями данных происходит аналогичным образом.

2. Разработка программных модулей в среде Python

2.1. Установки и настройка среды PyCharm

2.2. Особенности задания имен переменных и правила организации кода в Python

2.3. Работа с циклами, строками и списками

2.3.1 Выполнений задач на ветвление. (часть 2)

2.3.2 Отладка программного кода встроенными методами IDE (теория)

2.3.3 Выполнение задач на строки (часть 3)

2.3.4 Выполнение задач на списки (часть 4)

2.4. Именованные массивы – словари

2.4.1 Понятие и назначение словарей (теория)

2.4.2 Использование методов (часть 1)

2.4.2 Разработка программы по обработке коллекций (часть 2, 3)

2.4.3 Разработка интерфейса программы (часть 4)

2.5. Картежи. Работа с файлами в Python

2.5.1 Понятие картежей и их назначение (теория)

2.5.2 Основы работы с внешними файлами в Python (теория)

2.5.2 Добавление в программу функционала по работе с csv файлами (программа из п.2.4.2)

3. Разработка мобильных приложений.

3.1 Разработка дизайна мобильного приложения в среде Figma

3.2 Разработка класса на языке Kotlin

3.3 Работа в среде Android

3.3.1 Структура проекта Android

3.3.2 Создание проекта по заданию

4. Тестирование программных модулей

4.1. План тестирования (перечислить функции, которые будут тестироваться)

4.2. Сценарии тестирования с результатами (по шаблону)

4.3. Предложения по улучшению функциональности (фичи)

5. Разработка технической документации

5.1. Техническое задание

5.2. Руководство программиста

5.3. Руководство пользователя

Заключение

Список литературы