Министерство образования и науки Нижегородской области

Государственное бюджетное профессиональное образовательное учреждение

«Нижегородский Губернский колледж»

Допустить к защите:
преподаватели
_____ Е.П. Голубева

«__»____ _____2025 г.

ОТЧЁТ
ПО УЧЕБНОЙ ПРАКТИКЕ
модуль: ПМ. 01. Разработка модулей программного обеспечения для компьютерных систем

Руководитель_____Голубева Е.П. 01.03.2025 г.
Студент_____Астапчик Д.А. 01.03.2025 г.

Специальность, группа: 09.02.07, 41П

Нижний Новгород

2025 г.

СОДЕРЖАНИЕ

1. ПРОЕКТИРВОАНИЕ ПРОГРАММНОГО ПРОДУКТА

1.1 Проектирование и разработка БД

На данном этапе было необходимо произвести всесторонний анализ технического задания, что предоставил в последствии возможность произвести практичное проектирование и создание базы данных для хранения специализированной информации.

В ходе произведённого анализа предметной области и требований к программному продукту был выявлен перечень сущностей, необходимый для эффективного функционирования ИС “Управление мероприятиями” (EventManagement):

1. Страны (Countries) – предназначена для хранения информации о странах пользователей системы;
2. Статусы пользователей (User_states) – предназначена для хранения информации о статусах пользователей;
3. Гендеры (Genders) – предназначена для хранения информации о гендерах пользователей системы;
4. Города (Cities) – предназначена для хранения информации о города мероприятий;
5. Картинки городов (Images_city) – предназначена для хранения информации о путях к картинкам;
6. Картинки и города (Images_cities) – предназначена для хранения информации о картинках и городах;
7. Направления (Direction) – предназначена для хранения информации о направлениях;
8. События модераторов (Moderator_event) – предназначена для хранения информации об определённых событиях модераторов;
9. Пользователи (Users) – предназначена для хранения информации о всех пользователях системы;
10. Модераторы (Moderators) – предназначена для хранения информации о модераторах и направлениях, по которым они работают;

11. Жури (Jury) – предназначена для хранения информации о жури и направлениях, по которым они работают

12. Мероприятия (Events) – предназначена для хранения информации о мероприятиях;

13. Активности (Activies) – предназначена для хранения информации об активностях мероприятий.

В последствии был произведён процесс, целью которого являлась разработка базы данных, на основе составленных данных и заявленных требований технического задания.

Рассмотрим результат создания базы данных (рис 1-3).

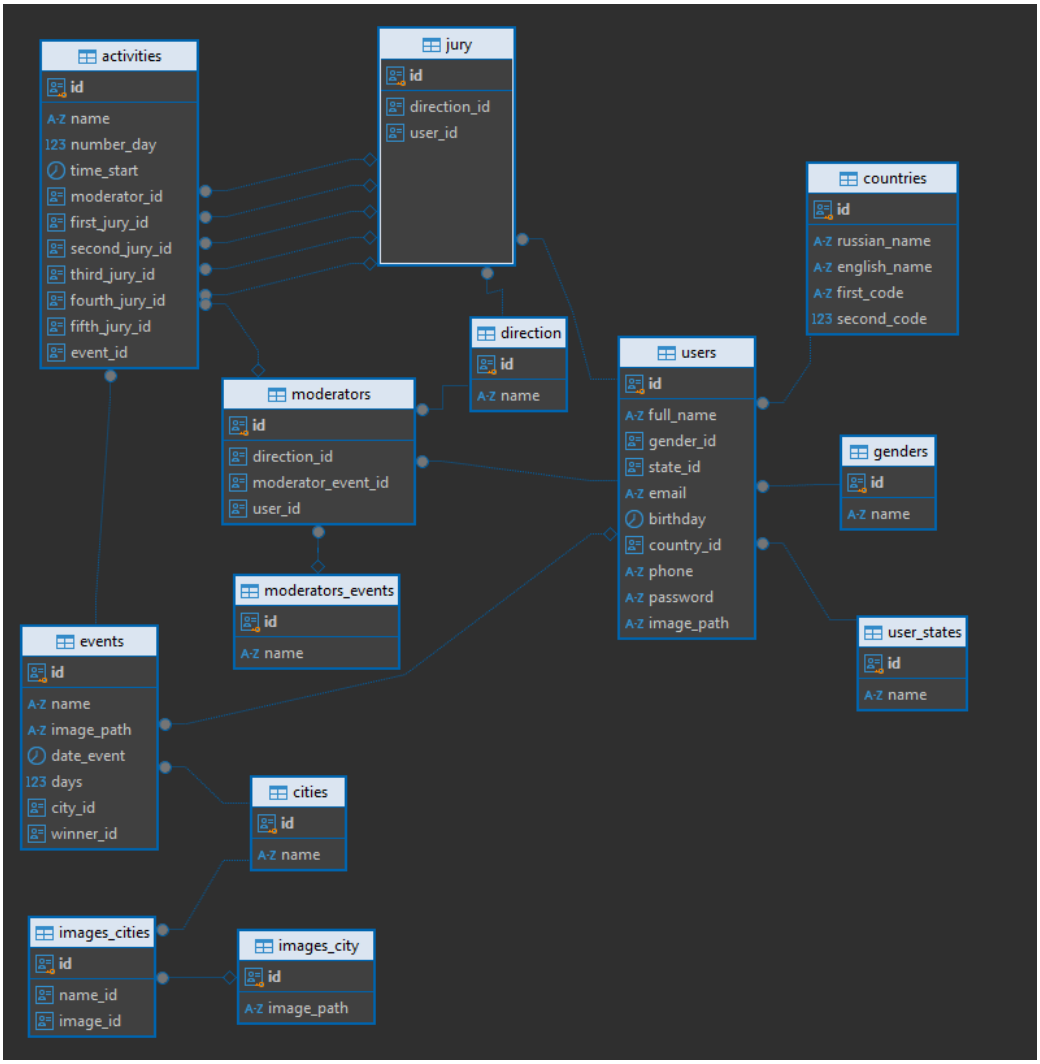

Рисунок 1 – ER-диаграмма БД “EventManagemet”

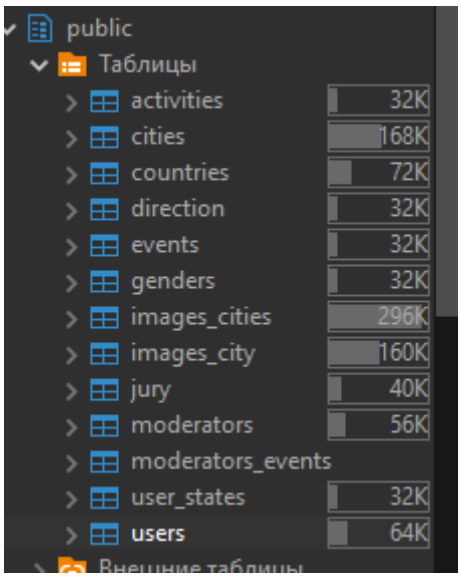

Рисунок 2 – Перечень сущностей БД

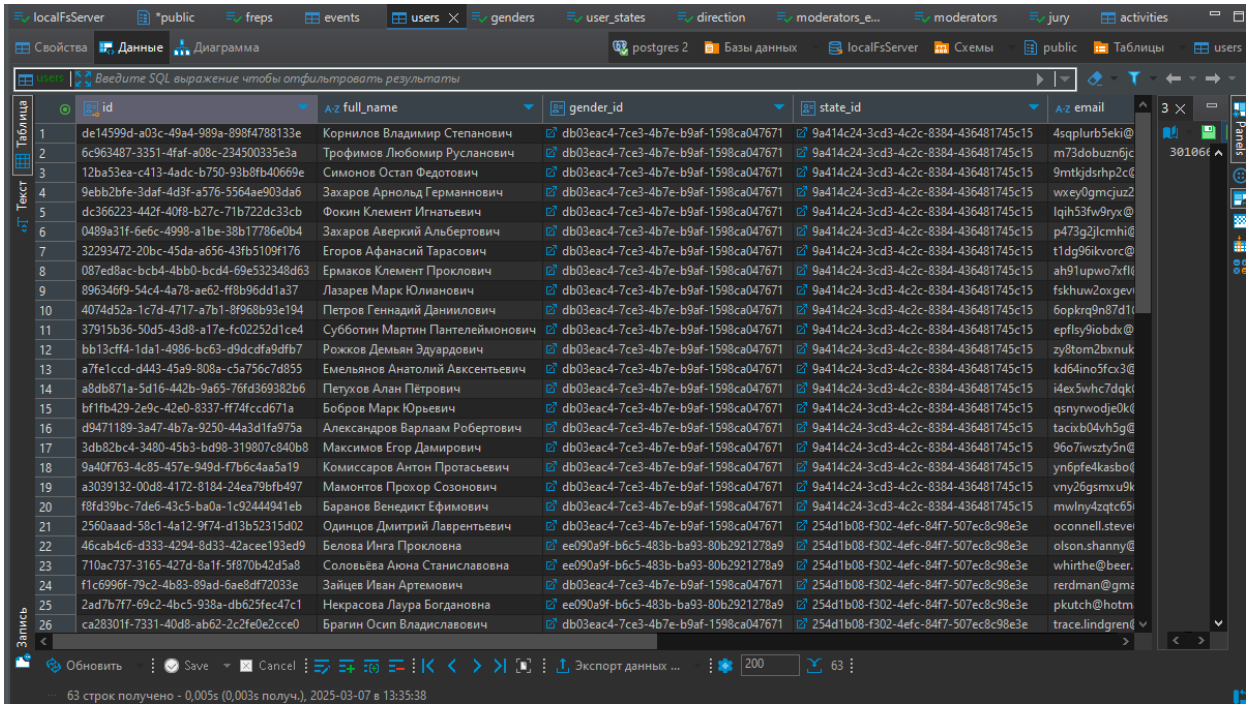

Рисунок 3 – Результат добавления данных в сущность Users

В случае необходимости, полноценно оценить структуру определённой сущности и данные загруженные в неё, смотреть DbScripts в репозитории проекта.

1.1 Разработка диаграммы прецедентов

На данном этапе для того, чтобы оценить все возможные взаимодействия пользователя с системой, посредствам визуального представления. Данная диаграмма предоставит возможность определить, какие функции доступны различным актёрам системы.

Рассмотрим результат создания диаграммы прецедентов (рис 4).

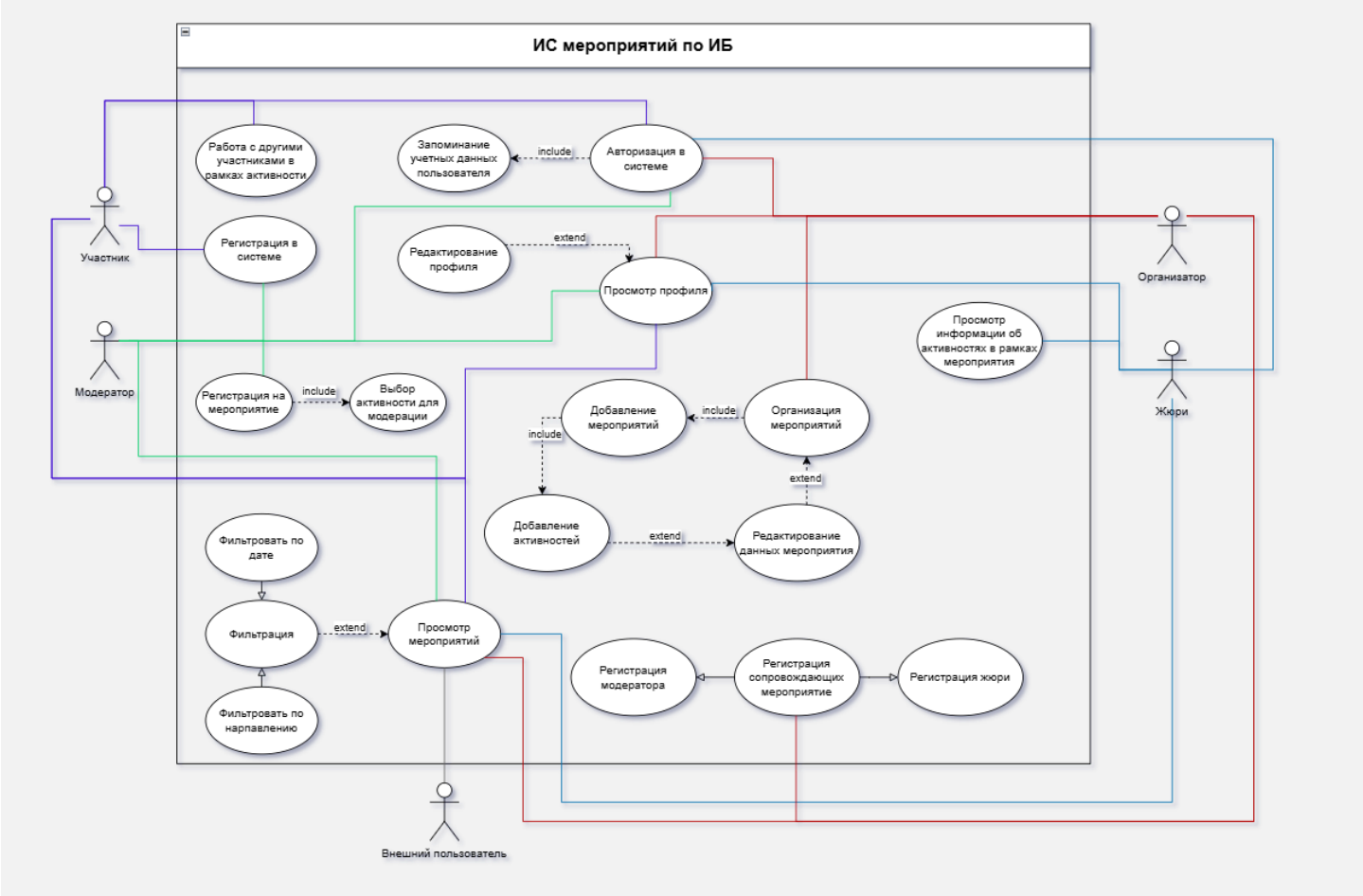

Рисунок 4 – Диаграмма прецедентов “EventManagements”

1.2 Разработка DataDictonary

На данном этапе было необходимо произвести разработку DataDictonary (словарь данных), что предназначен для структурирования базы данных, посредствам которого будет полностью отображено подробное описание всех данных, ограничений, типов и т.п используемое в сущностях БД.

Рассмотрим результат создания словаря данных (рис 5,6).

96	Activities				
97	KEY	FIELD NAME	DATA TYPE / FIELD SIZE	REQUIRED ?	NOTES
98	PK	id	uuid	Y	gen_random_uuid()
99		name	varchar	Y	
100		number_day	int4	Y	Номер дня начала активности
101		time_start	time	Y	Время начала активности
102	FK	moderator_id	uuid	N	Внешний ключ от сущности Moderators
103	FK	first_jury_id	uuid	N	Внешний ключ от сущности Jury
104	FK	second_jury_id	uuid	N	Внешний ключ от сущности Jury
105	FK	third_jury_id	uuid	N	Внешний ключ от сущности Jury
106	FK	fourth_jury_id	uuid	N	Внешний ключ от сущности Jury
107	FK	fifth_jury_id	uuid	N	Внешний ключ от сущности Jury
108	FK	event_id	uuid	Y	Внешний ключ от сущности Events
109					
110					

Рисунок 5 – Сущность “Активности”

4	Countries				
5	KEY	FIELD NAME	DATA TYPE / FIELD SIZE	REQUIRED ?	NOTES
6	PK	id	uuid	Y	gen_random_uuid()
7		russian_name	varchar	Y	
8		english_name	varchar	Y	
9		first_code	varchar	Y	Код, сотоящий из букв латинского алфавита, что индентифицирует страну
10		second_code	int	Y	Код, состоящий из цифр, что идентифицирует страну
11					

Рисунок 6 – Сущность “Страны”

В случае необходимости, полноценно оценить словарь данных, смотреть docs/ DataDictonary в репозитории проекта.

1.3 Технологии разработки программного обеспечения

На данном этапе был произведён процесс проектирования программного продукта, что заключался в выборе специализированных технологий и паттернов для разработки программного продукта.

В результате всестороннего анализа технического задания, нынешних технологий было выявлено, что для разработки программного продукта соответствующего требования технического задания необходимо использовать паттерн MVVM.

Для написания программного кода, в областях логики ViewModels и Models, будет использоваться интегрированная среда разработки VisualStudio с языком программирования C#.

Для того, чтобы реализовать интеграцию БД, будет использованы технологии EntityFramework и Linq запросов, что предоставят возможность эффективно выполнять различные операции с созданной БД.

Аналогично, с целью произвести реализацию графического интерфейса приложения будет использован фреймворк Avalonia.

1.4 Проектирование дизайна приложения

На данном этапе проектирования программного продукта был составлен примерный дизайн приложения, имеющий специализированную цветовую гамму и формат фигур. Рассмотрим результат (рис 7).

Рисунок 7 – Примерный дизайн приложения “Управления мероприятиями”

2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

На данном этапе было необходимо реализовать цели, что заключались в разработке десктопного приложения, имеющего графический интерфейс, что соответствует определённым требованиям, а также разработке DLL специализированного направления. Аналогично было необходимо произвести юнит-тестирование разработанной DLL.

2.1 Интеграция базы данных в приложение

На данном этапе был произведён процесс, целью которого является интеграция базы данных в программный продукт. Аналогично в последствии был разработан модуль с запросами к БД для того, чтобы реализовать практичную работу с определёнными данными в рамках программного решения.

Перечень видов запросов реализованных к базе данных:

1. Select (Выбрать) – запросы на получение данных из БД, что необходимы для реализации работы программного продукта, соответствующего требованиям технического задания;
2. Update (Обновить) – запросы на обновление данных в БД, что необходимы для реализации работы программного продукта, соответствующего требованиям технического задания;
3. Insert (Вставить) – запросы на добавление данных в БД, что необходимы для реализации работы программного продукта, соответствующего требованиям технического задания.

Рассмотрим структуру базы данных, что была интегрирована в программный продукт (рис 8).

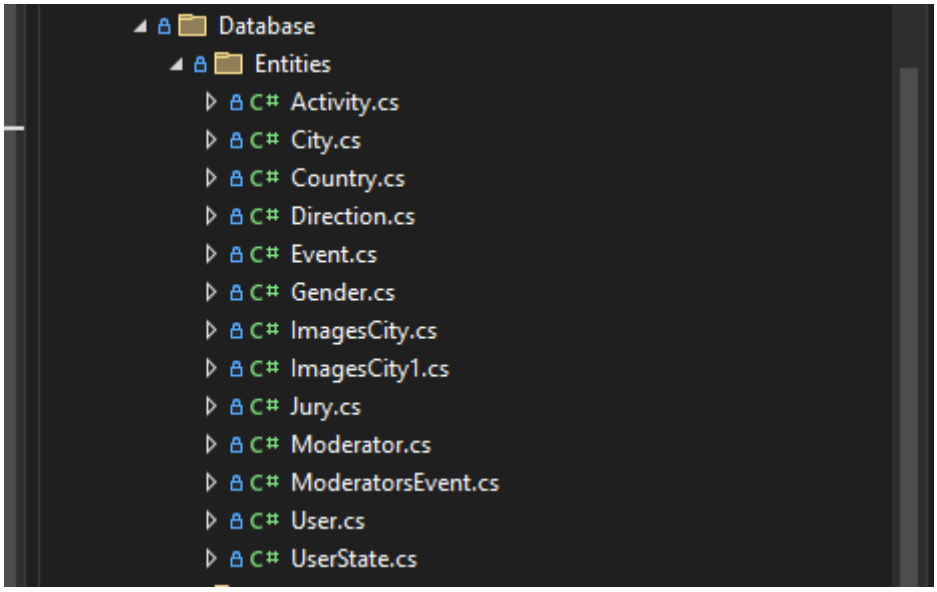

Рисунок 8 – Структура БД в Visual Studio

В последствии был произведён процесс, целью которого являлась реализация запросов к БД, перечень, которых был ранее рассмотрен. В ходе разработки запросов, была составлена следующая структура (рис 9).

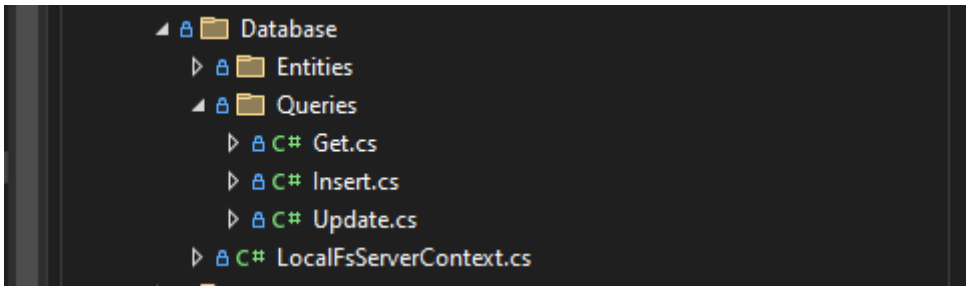

Рисунок 9 – Структура реализованных запросов

В качестве примера, рассмотрим некоторых из разработанных запрос, для демонстрации результата произведённой деятельности (рис 10, 11, 12).

```
/// <summary>
/// Запрос на получение всех пользователей и сопутствующей информации
/// </summary>
/// <returns>Список всех пользователей</returns>
Ссылок: 1
public static List<User>? AllUsers()
{
    LocalFsServerContext db = new();
    List<User>? users = null;
    try
    {
        users = db.Users
            .Include(p=>p.Gender)
            .Include(p=>p.State)
            .Include(p => p.Country)
            .ToList();
    }
    catch { }

    return users;
}
```

Рисунок 10 – Запрос на получение всех пользователей (Get.AllUsers)

Данный запрос предоставляет возможность получить всю информацию о пользователях, включая их пол или статус или страну, в рамках разрабатываемой системы и предоставленной предметной области. В последствии данная информация будет использована для определения пользователя, в процессе прохождения авторизации.

```
/// <summary>
/// Запрос на добавление нового пользователя в систему
/// </summary>
/// <param name="user">Объект пользователя</param>
/// <returns>Успешно/неуспешно</returns>
Ссылок: 1
public static bool User(User user)
{
    LocalFsServerContext db = new();
    bool isSaved = true;

    try
    {
        _ = db.Users.Add(user);
        _ = db.SaveChanges();
    }
    catch { isSaved = false; }

    return isSaved;
}
```

Рисунок 11 – Запрос на добавление записи о новом пользователе (Insert.User)

Данный запрос предоставляет возможность добавить нового пользователя в систему. В последствии данный запрос будет использоваться при регистрации организатором жюри/модератора в системе.

```
/// <summary>
/// Запрос на обновление данных пользователя
/// </summary>
/// <param name="user">Объект пользователя</param>
/// <returns>Успешно/неуспешно</returns>
Ссылок: 1
public static bool DataUser(User user)
{
    bool isSaved = true;
    LocalFsServerContext db = new();
    User? loadRecord = null;
    try
    {
        loadRecord = db.Users
            .FirstOrDefault(p => p.Id == user.Id);
        if (loadRecord != null)
        {
            loadRecord.Email = user.Email;
            loadRecord.Password = user.Password;
            loadRecord.Phone = user.Phone;
            _ = db.SaveChanges();
        }
        else
        {
            throw new Exception("Не найдена запись");
        }

    }
    catch { isSaved = false; }
    return isSaved;
}
```

Рисунок 12 – Запрос на обновление данных о пользователе (Update.DataUser)

Данный запрос предоставляет возможность обновить определённую информацию о пользователе в базе данных. В последствии данный запрос был использован в целях реализации окна редактирования профиля определённого пользователя системы.

2.2 Сохранения пользователя в системе

На данном этапе был произведён процесс, целью которого являлась разработка механизма для сохранения пользователя в системе, в результате успешной авторизации.

Данный механизм был реализован посредствам сохранения специализированного токена пользователя в системе. Сохранение осуществляется в определённый .json файл, что будет храниться в системном каталоге персонального компьютера пользователя. При запуске приложения, на этапе навигации, будет осуществляться проверка, целью которой является удостоверить существование сохранённого токена. В случае если файла, содержащего токен (или самого токена) не существует, пользователю будет необходимо пройти авторизацию.

Рассмотрим некоторых из методов, предоставляющие возможность практично, реализовать данную задачу (рис 13, 14).

```
Ссылок: 3
static string GetFilePath()
{
    //Получаем путь к папке appData, SpecialFolder - перечсиление содержащее системные пути к папкам
    string appData = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);

    //Формируем путь к каталогу EventManagemetn
    string appFolder = Path.Combine(appData, "EventManagemetn");
    if (!Directory.Exists(appFolder))
    {
        Directory.CreateDirectory(appFolder); //Если его не существует создаём, данный каталог по пути
    }

    //Возвращаем сформированный путь к файлу Data.json, что ещё не был создан
    return Path.Combine(appFolder, "Data.json");
}
```

Рисунок 13 – Получение пути к файлу, хранящему токен пользователя

Данный метод предназначен для того, чтобы получить путь к файлу, где будет храниться токен пользователя. В случае если каталог содержащий файл токена не будет существовать, будет создан новый с наименованием “EventManagement”.

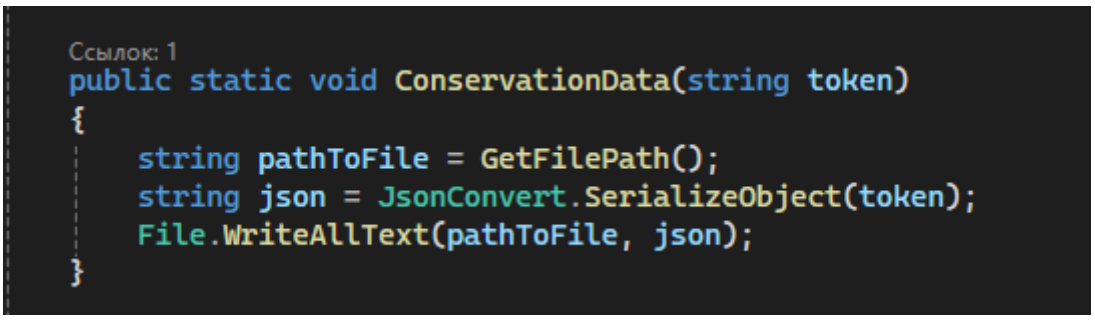

Рисунок 14 – Сохранение токена в системе

Данный метод предназначен для сохранения токена пользователя по определённому пути в системе. В комбинации с другими методами класса ConservationDataAuthorizedUser.cs будет успешно реализован механизм сохранения данных пользователя в результате прохождения авторизации пользователем.

В случае необходимости, полноценно оценить функционал механизма сохранения пользователя в системе после его авторизации, смотреть проект приложения Models/ ConservationDataAuthorizedUser.cs в репозитории проекта.

2.3 Реализация капчи

На данном этапе было необходимо произвести процесс, целью которого являлась разработка изображения капчи, что будет использоваться при авторизации пользователя в системе. Для реализации данной задачи были разработаны методы, что генерируют случайное значение, состоящее из 4 символов, строчных/прописных букв, цифр и генерируют специализированное изображение, с визуальным шумом и сгенерированным значением captcha.

С целью избежать избыточности материала, в качестве иллюстрирующего материала будет рассмотрен метод для генерации необходимого значения captcha (рис 15). В случае необходимости полноценно произвести анализ данного функционала, смотреть репозиторий проекта Models/Captcha/CaptchaGenereate.cs.

```
/// <summary>
/// Метод предназначенный для генерации текста капчи
/// </summary>
/// <returns></returns>
Ссылок: 1
public string GenerateCaptchaText()
{
    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYabcdefghijklmnopqrstuvwxyzZ0123456789";
    return new string(Enumerable.Repeat(chars, 4)
        .Select(s => s[random.Next(s.Length)]).ToArray());
}
```

Рисунок 15 – Генерация значения captcha

2.4 Разработка графического интерфейса и его бизнес-логики

На данном этапе был произведён основной процесс, что заключался в разработке графического интерфейса приложения (View) и его бизнес-логи (ViewModels).

Для того, чтобы полноценное реализовать данную задачу, было необходимо произвести процесс следующего перечня окон (View, ViewModel):

1. Окно мероприятий;
2. Окно пользователя (окно организатора, жюри, участника);
3. Окно профиля пользователя;
4. Окно редактирования профиля пользователя;
5. Окно, содержащее информацию о всех жюри системы;
6. Окно, содержащее информацию о всех участниках системы;
7. Окно регистрации жюри/модератора.

Для того, чтобы избежать избыточности материала, в качестве примера будут рассмотрены окна мероприятий, организатора, окно регистрации жюри/модератора. В случае необходимости полноценно проанализировать содержимое проекта смотреть репозиторий проекта ViewModels, View.

Окна будут рассматривать в ранее указанном порядке. Рассмотрим результат произведённой деятельности (рис 16-).

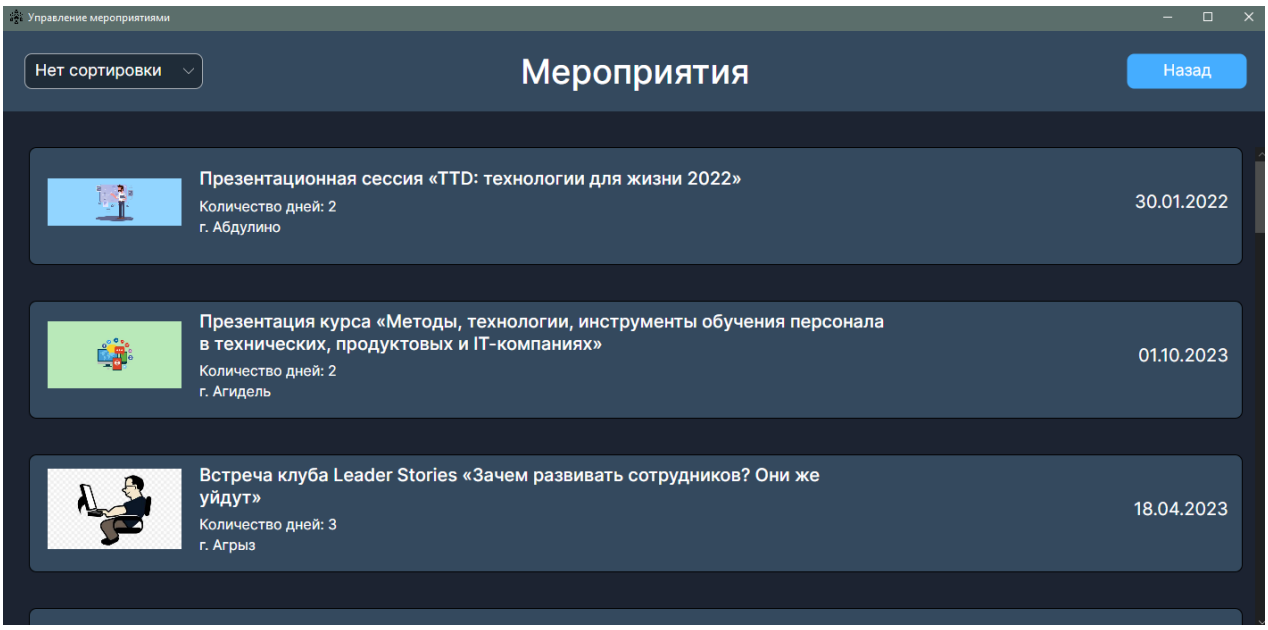

Рисунок 16 – Окно мероприятий (MainView.axaml)

На данном окне (рис 16), демонстрируется информация о всех мероприятия, что были импортированы в базу данных. Аналогично у пользователя системы присутствует возможность произвести фильтрацию мероприятий по количеству дней мероприятия и дате мероприятия. Указанный функционал реализуется в MainViewModel.cs, рассмотрим содержимое данного файла (рис 17).

```
Ссылок: 1
void OrderEventsBy()
{
    if (Events != null)
    {
        if (SelectedOption == "По дате")
            Events = Events.OrderByDescending(p => p.DateEvent).ToList();
        else if (SelectedOption == "По длительности")
            Events = Events.OrderByDescending(p => p.Days).ToList();
        else
            Events = new List<Event>(EventsFixed);
    }
}
```

Рисунок 17 – Метод фильтрации выводимого списка (MainViewMode.OrderEventsBy)

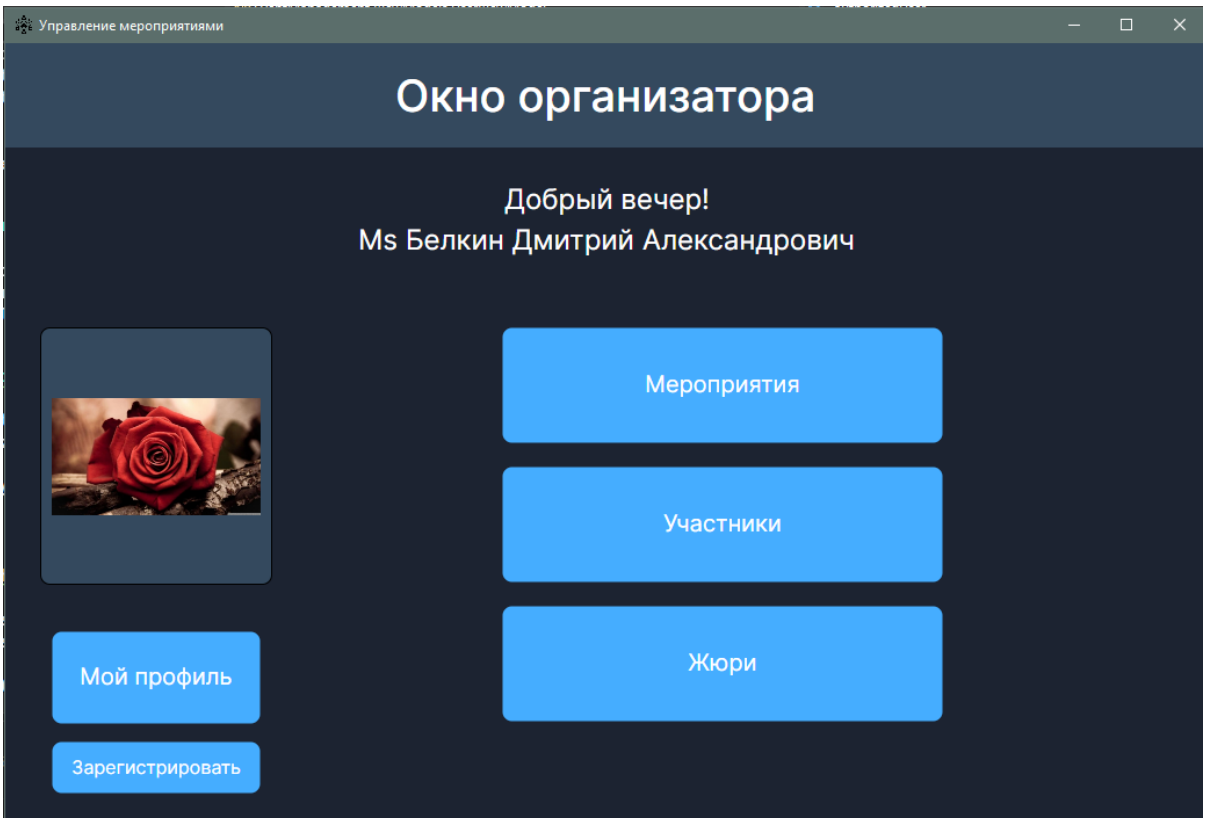

Рисунок 18 – Окно пользователя (UserView.axaml)

Данное окно (рис 18), является некоторой начальной точкой, для перехода к остальным окнам программного продукта. Организатор с окна пользователя может перейти в окно с списком всех мероприятий, в окно с списком всех участников и в окно с списком всех жюри. Аналогично, после авторизации, при переходе на данное окно, осуществляются процессы, что определяют, как на данный момент период дня, и кто перешёл на данное окно. У организатора есть возможность перейти в окно со своим профилем, где будет продемонстрирована его персональная информация и предоставлена возможность отредактировать профиль, выйти из аккаунта, вернуться к окну пользователя. Также организатор имеет возможность перейти на окно регистрации жюри/модератора посредствам нажатия на кнопку “Зарегистрировать”.

Рассмотрим некоторые из функций, реализующихся на данном окне (рис 19, 20).

```
/// <summary>
/// Определение текущего периода времени
/// </summary>
/// <returns>Текущий период времени</returns>
Ссылок: 1
public static string DefiningPeriodDay()
{
    int hour = DateTime.Now.Hour;
    int minute = DateTime.Now.Minute;

    if (hour < 11 || (hour == 11 && minute == 0))
        return "Доброе утро!";
    else if ((hour == 11 && minute >= 1) || hour < 18)
        return "Добрый день!";
    else
        return "Добрый вечер!";
}
```

Рисунок 19 – Определение текущего периода времени (ProcessingDataForUser.DefiningPeriodDay)

Данный метод предоставляет возможность по времени установленному в системе определить, какой на данный момент период времени, чтобы вернуть определённой значение в представление на вывод для пользователя.

```
    Ссылок: 0
    public void GoToMainView() => MainWindowViewModel.Navigation.NavigateToMainViewEvents();
    Ссылок: 0
    public void GoToProfileView() => MainWindowViewModel.Navigation.NavigateToProfileView();
    Ссылок: 0
    public void GoToJuryView() => MainWindowViewModel.Navigation.NavigateToJuryView();
    Ссылок: 0
    public void GoToParticipantsView() => MainWindowViewModel.Navigation.NavigateToParticipantsView();

    Ссылок: 0
    public void GoToRegistrationView() => MainWindowViewModel.Navigation.NavigateToRegistrationView();
}
```

Рисунок 20 – Список навигационных методов (UserViewModels.cs)

На указанном иллюстрирующем материале (рис 20) продемонстрирован список навигационных методов, что используются в качестве команды для кнопок, посредствам на нажатие которых осуществляется переход на различные окна приложенининия.

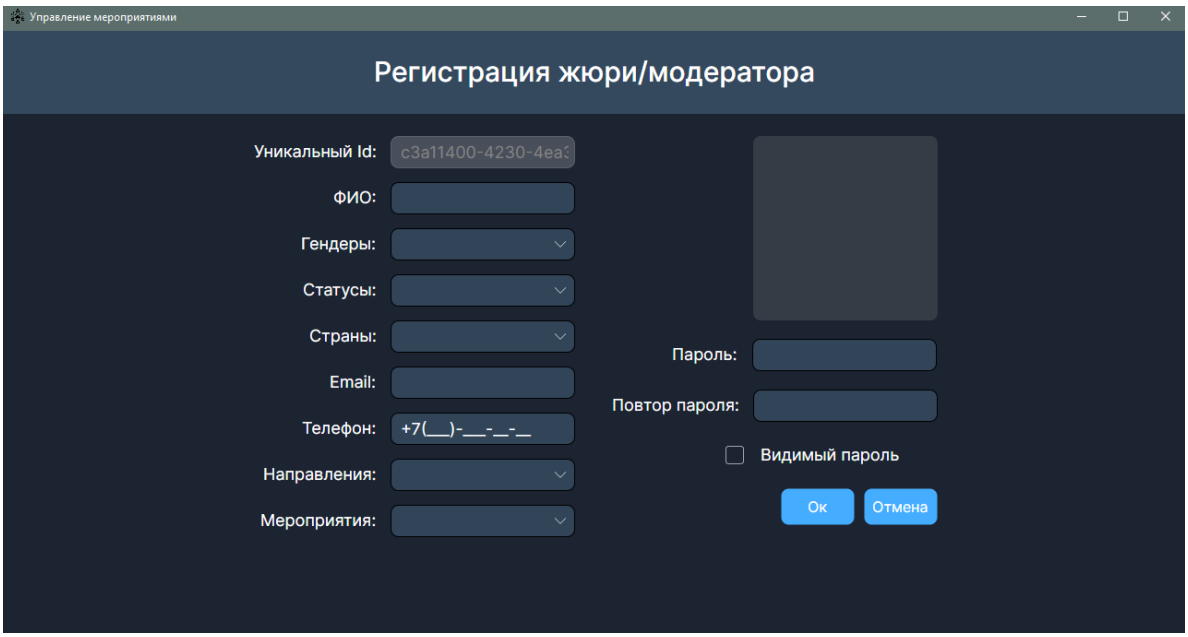

Рисунок 21 – Окно регистрации жюри/модератора (RegistrationView.axaml)

На представленном иллюстрирующем материале (рис 21) предоставлена форма для заполнения данных о новом модераторе или жюри. Уникальный Id генерируется автоматически при запуске данного окна. Была добавлена маска на ввод телефона. Присутствует возможность включить видимость пароля, посредствам CheckBox. Присутствуют различный перечень проверок, рассмотрим некоторые из них (рис 21, 22): проверка на валидность почты, проверка на валидность пароля. Аналогична, был реализован функционал добавления картинок (рис 23).

Рассмотрим результат осуществлённой деятельности.

```
/// <summary>
/// Проверка на корректность шаблона почты
/// </summary>
/// <returns></returns>
Ссылок: 1
bool IsEmailValid()
{
    // Регулярное выражение для проверки email
    string emailPattern = @"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";

    return Regex.IsMatch(Email, emailPattern);
}
```

Рисунок 21 – Проверка почты на валидность (RegistrationViewModel.cs)

```
/// <summary>
/// Проверка на валидность пароля
/// </summary>
/// <returns></returns>
Ссылок: 1
bool IsPasswordValid()
{
    // Проверяем длину пароля
    if (Password.Length < 6)
        return false;

    // Проверяем наличие хотя бы одной заглавной буквы, строчной буквы, цифры и спецсимвола
    bool hasUpperCase = false;
    bool hasLowerCase = false;
    bool hasDigit = false;
    bool hasSpecialChar = false;

    foreach (char c in Password)
    {
        if (char.IsUpper(c)) hasUpperCase = true;
        if (char.IsLower(c)) hasLowerCase = true;
        if (char.IsDigit(c)) hasDigit = true;
        if (Regex.IsMatch(c.ToString(), @"[\W_]")) hasSpecialChar = true; // Спецсимволы
    }

    return hasUpperCase && hasLowerCase && hasDigit && hasSpecialChar;
}
```

Рисунок 22 – Проверка пароля на валидность (RegistrationViewModel.cs)

```
/// <summary>
/// Выбор картинки
/// </summary>
/// <returns></returns>
/// <exception cref="NullReferenceException"></exception>
Ссылок: 0
public async Task SelectImageAsync()
{
    try
    {
        if (Application.Current?.ApplicationLifetime is not IClassicDesktopStyleApplicationLifetime desktop ||
        desktop.MainWindow?.StorageProvider is not { } provider) throw new NullReferenceException("Отсутствует провайдер");

        var files = await provider.OpenFilePickerAsync(
            new FilePickerOpenOptions()
            {
                Title = "Выберите изображение профиля",
                AllowMultiple = false,
                FileTypeFilter = [FilePickerFileTypes.ImageAll]
            });

        if (files is null || files.Count == 0)
            return;

        SelectedFile = files[0];

        Uri fileUri = SelectedFile.Path;
        string pathFile = fileUri.LocalPath;

        SelectedImage = new Bitmap(pathFile);
    }
    catch { }
}
```

Рисунок 23 – Добавление картинки (RegistrationViewModel.cs)

Данный метод предоставляет возможность организатору добавить картинку новому пользователю (жюри/модератор) в представление RegistrationView.axaml. Данная задача осуществляется посредствам того, что сначала проверяется доступность провайдера хранения, в случае если нет доступа, производится исключение. Иначе, открывается окно выбора файла, где в последствии по полученному файлу, можно получить необходимый путь к картинке и загрузить картинку в определённый элемент разметки графического интерфейса приложения.

В случае возникновения необходимости полноценно проанализировать разработанный функционал, смотреть репозиторий проекта ViewModels/ RegistrationViewModel.cs.

2.5 Разработка DLL, unit-testing

На данном этапе выполнения предоставленной работы, было необходимо реализовать процесс, целью которого является разработка специализированной DLL и её дальнейшее тестирование.

Рассмотрим функционал, реализованный в DLL:

1. Boolean CheckMark(String mark): данный метод проверяет переданный номерной знак в формате a999aa999 (латинскими буквами) и возвращает true или false в зависимости от правильности номерного знака. Метод должен учитывать также и существующие номера регионов;
2. String GetNextMarkAfter(String mark): данный метод принимает номерной знак в формате a999aa999 (латинскими буквами) и выдает следующий номер в данной серии или создает следующую серию;
3. String GetNextMarkAfterInRange(String prevMark, String rangeStart, String rangeEnd): данный метод принимает номерной знак в формате a999aa999 (латинскими буквами) и выдает следующий номер в данной данном промежутке номеров rangeStart до rangeEnd (включая обе границы). Если нет возможности выдать следующий номер, необходимо вернуть сообщение “out of stock”;
4. int GetCombinationsCountInRange(String mark1, String mark2): данный метод принимает два номера в формате a999aa999 (латинскими буквами) и возвращает количество возможных номеров между ними (включая обе границы). Метод необходим, чтобы рассчитать оставшиеся свободные номера для региона.

В качестве иллюстрирующего материала будет рассмотрен метод, определяющий количество возможных номеров между двумя заданными номерами (рис 24).

```
/// <summary>
/// Метод, принимает два номера и определяет количество номеров между (включая границы)
/// </summary>
/// <param name="mark1">Номер 1</param>
/// <param name="mark2">Номер 2</param>
/// <returns>Количетсво возможных номер в диапазоне</returns>
Ссылок: 3
public int GetCombinationsCountInRange(String mark1, String mark2)
{
    if (CheckMark(mark1) && CheckMark(mark2))
    {
        mark1 = EnglishToRussianChar(mark1);
        mark2 = EnglishToRussianChar(mark2);
        int counter = 0;
        string start = mark1;
        MarkReg startRegMark = new MarkReg(mark1);
        MarkReg endRegMark = new MarkReg(mark2);
        while (startRegMark <= endRegMark)
        {
            start = GetNextMarkAfter(start);
            startRegMark = new MarkReg(start);
            counter++;
        }
        return counter;
    }
    return 0;
}
```

Рисунок 24 – Определение количества возможных номеров в заданном диапазоне

В последствии был произведён процесс разработки юнит-тестов, с целью дальнейшего тестирования DLL. Рассмотрим результат осуществлённого тестирования (рис 25).

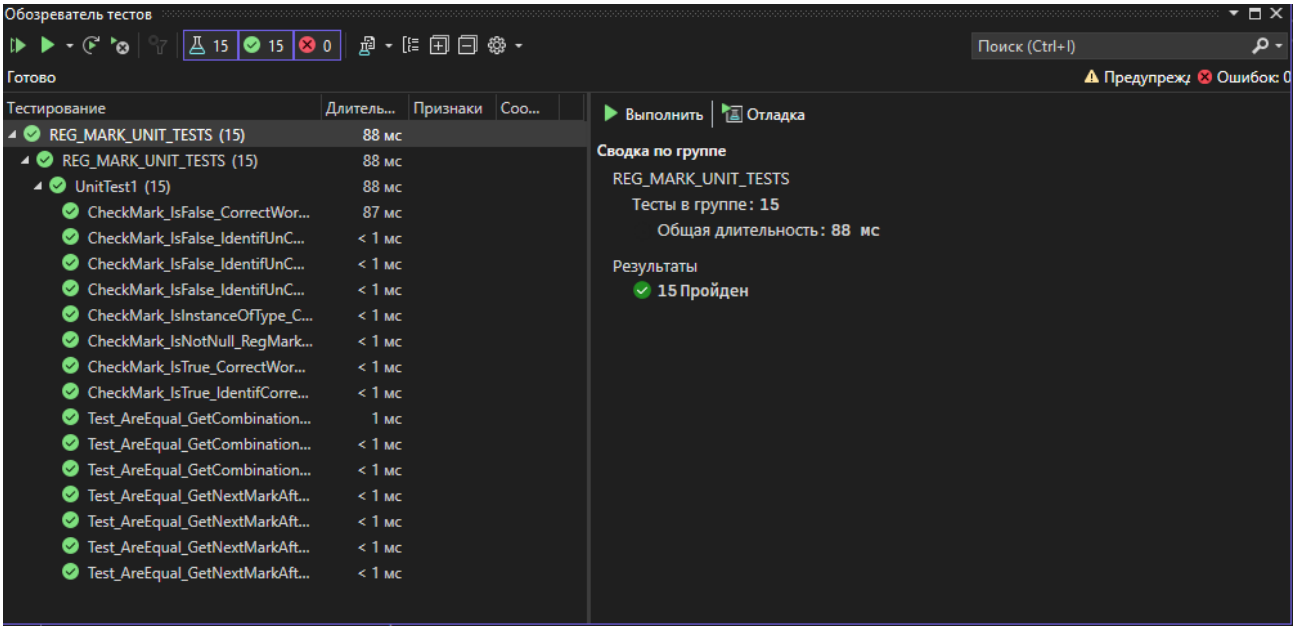

Рисунок 25 – Результат юнит-тестирования

ВЫВОД

В результате выполнения предоставленной работы был успешно разработан программный продукт "Управление мероприятиями", предназначенный для автоматизации процессов регистрации, организации и проведения мероприятий. Приложение предоставляет удобный интерфейс для различных категорий пользователей (организаторов, модераторов, жюри и участников), обеспечивая гибкую систему доступа и управления.

Разработанная система предоставляет возможности:

1. Просматривать список мероприятий с возможностью фильтрации по дате и длительности;
2. Управлять пользователями и их ролями (регистрация модераторов и жюри);
3. Обеспечивать авторизацию пользователей с защитой через CAPTCHA;
4. Проверять корректность вводимых данных (валидность почты, пароля и изображений);
5. Сохранять пользовательскую сессию после авторизации;
6. Позволять пользователям редактировать свой профиль.

Приложение обеспечивает оптимизацию и автоматизацию процессов управления мероприятиями, исключая необходимость ведения ручного учета данных, повышая скорость обработки информации и снижая вероятность ошибок.

Разработанный продукт представляет собой полноценную и готовую к использованию систему, которая может быть применена в сфере организации мероприятий.

Дополнительно была разработана специализированная DLL для работы с регистрационными номерами автомобилей, что расширяет функциональные возможности системы. Для повышения надежности кода было проведено юнит-тестирование методов данной библиотеки, что подтвердило корректность её работы.