ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО
Научный руководитель,
доцент департамента
Программной инженерии

_____Р.З. Ахметсафина
«___» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»

_____В. В. Шилов
«___» _____ 2020 г.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ “VOBSHAGE”

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.03-01 12 01-1-ЛУ

Исполнитель

студент группы БПИ197

_____ /Бакытбек уулу Н. /

«___ »_____ 2020 г.

Москва 2020

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.03-01 12 01-1				

УТВЕРЖДЕН
RU.17701729.04.03-01 12 01-1

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.04.03-01 12 01-1

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ “VOBSHAGE”

Текст программы

RU.17701729.04.03-01 12 01-1

Листов 67

Москва 2020

Оглавление

1 Проект appUI 3
1.1 DataModels 3
1.1.1 CheckList.cs 3
1.1.2 Team.cs 6
1.1.3 User.cs 8
1.2 Models 10
1.2.1 LocalNotification.cs 10
1.2.2 TaskItem.cs 11
1.2.3 TaskPageModel.cs 13
1.3 pages 17
1.3.1 AuthPage.cs 17
1.3.2 CheckListPage.cs 20
1.3.3 GeneralPage.cs 26
1.3.4 HelpPage.cs 33
1.3.5 ModalTasks.cs 34
1.3.6 PersonalPage.cs 40
1.3.7 TeamsPage.cs 45
2 Проект appUI.Anroid 52
2.1 Listeners 52
2.1.1 AppDataHelper.cs 52
2.1.2 CheckListListener.cs 53
2.2 Persistents 55
2.2.1 LocalNotificationService.cs 55
2.2.2 MyFirebaseAuth.cs 61
2.2.3 SQLiteDb.cs 63
Список используемой литературы 64
Лист регистрации изменений. 65

1 Проект appUI

1.1 DataModels

1.1.1 CheckList.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using appUI.Models;
using System.Collections.ObjectModel;
using Newtonsoft.Json;

namespace appUI.DataModels
{
    /// <summary>
    /// Модель чек-листа.
    /// </summary>
    public class CheckList : INotifyPropertyChanged
    {
        /// <summary>
        /// Оповещает об изменеии свойств.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// Название узла в онлайн базе данных Firebase database.
        /// </summary>
        public static readonly string resourseName = "CheckLists";
        /// <summary>
        /// Заголовок чек-листа.
        /// </summary>
        string title;
        /// <summary>
        /// Идентификатор чек-листа.
        /// </summary>
        string id;
        /// <summary>
        /// Дата чек-лиса.
        /// </summary>
        DateTime date;
        /// <summary>
        /// Время чек-листа.
        /// </summary>
        TimeSpan time;
        /// <summary>
        /// Свойство доступа к title. Оповещает об изменениях.
        /// </summary>
        public string Title {
            get => title;
```

```
        set {
            if(value != title)
            {
                title = value;
                SetProperty();
            }
        }
    }
    /// <summary>
    /// Свойство доступа к id. Оповещает об изменениях.
    /// </summary>
    public string Id {
        get => id;
        set {
            if (value != id)
            {
                id = value;
                SetProperty();
            }
        }
    }
    /// <summary>
    /// Свойство доступа к date. Оповещает об изменениях.
    /// </summary>
    public DateTime Date {
        get => date;
        set {
            if (value != date)
            {
                date = value;
                SetProperty();
            }
        }
    }
    /// <summary>
    /// Свойство доступа к time. Оповещает об изменениях.
    /// </summary>
    public TimeSpan Time {
        get => time;
        set {
            if (value != time)
            {
                time = value;
                SetProperty();
            }
        }
    }
    /// <summary>
    /// Задачи чек-листа. Оповещает об изменениях.
    /// </summary>
    public ObservableCollection<TaskItem> Tasks { get; set; }
    /// <summary>
```

```
        /// Флаг уведомления.
        /// </summary>
        bool isRemind;
        /// <summary>
        /// Свойство доступа к isRemind. Оповещает об изменениях.
        /// </summary>
        [JsonIgnore]
        public bool IsRemind
        {
            get => isRemind;
            set {
                if (value != isRemind)
                {
                    isRemind = value;
                    SetProperty();
                    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(RemindIcon)));
                }
            }
        }
        /// <summary>
        /// Возвращает картинку в зависимости от состояния уведомлений.
        /// </summary>
        [JsonIgnore]
        public string RemindIcon
        {
            get => isRemind ? "bell.png" : "nbell.png";
        }
        /// <summary>
        /// Идентификатор уведомлений.
        /// </summary>
        public int? NotificfId { get; set; }
        /// <summary>
        /// Оповещение об изменеии определенного свойства.
        /// </summary>
        /// <param name="name">Имя свойства.</param>
        protected void SetProperty([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
            //await firebase.Child(resourseName).Child(Id).PutAsync(this);

        }
    }
}
```

1.1.2 Team.cs

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace appUI.DataModels
{
    /// <summary>
    /// Модель команды.
    /// </summary>
    public class Team : INotifyPropertyChanged
    {
        /// <summary>
        /// Название узла в онлайн базе данных Firebase database.
        /// </summary>
        public static readonly string resourseName = "Teams";
        /// <summary>
        /// Идентификатор команды.
        /// </summary>
        string id;
        /// <summary>
        /// Имя команды.
        /// </summary>
        string name;
        /// <summary>
        /// Идентификатор-токены членов команды.
        /// </summary>
        public List<string> MembersToken { get; set; }
        /// <summary>
        /// Идентификатор чек-листов команды.
        /// </summary>
        public List<string> CheckListsId { get; set; }
        /// <summary>
        /// Свойство для доступа name. Оповещает об изменениях.
        /// </summary>
        public string Name
        {
            get => name;
            set {
                if (value != name)
                {
                    name = value;
                    SetProperty();
                }
            }
        }
        /// <summary>
        /// Свойство для доступа к id. Оповещает об изменениях.
```

```
        /// </summary>
        public string Id
        {
            get => id;
            set {
                if (value != id)
                {
                    id = value;
                    SetProperty();
                }
            }
        }
        /// <summary>
        /// Оповещает об изменеии свойств. Оповещает об изменениях.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        protected void SetProperty([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

1.1.3 User.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace appUI.DataModels
{
    /// <summary>
    /// Модель пользоветля.
    /// </summary>
    [Serializable]
    public class User : INotifyPropertyChanged
    {
        /// <summary>
        /// Оповещает об изменеии свойств.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// Название узла в базе данных Firebase Realtime Database.
        /// </summary>
        public static readonly string resourseName = "Users";
        /// <summary>
        /// Идентификатор пользователя.
        /// </summary>
        string id;
        /// <summary>
        /// Имя пользователя.
        /// </summary>
        string name;
        /// <summary>
        /// Идентификаторы команд пользователя.
        /// </summary>
        public List<string> TeamsId { get; set; }
        /// <summary>
        /// Свойство для доступа к name. Оповещает об изменениях.
        /// </summary>
        public string Name
        {
            get => name;
            set {
                if (value != name)
                {
                    name = value;
                    SetProperty();
                }
            }
        }
```

```
        /// <summary>
        /// Токен пользователя.
        /// </summary>
        public string Token { get; set; }
        /// <summary>
        /// Свойство для доступа к id. Оповещает об изменениях.
        /// </summary>
        public string Id
        {
            get => id;
            set {
                if (value != id)
                {
                    id = value;
                    SetProperty();
                }
            }
        }
        /// <summary>
        /// Оповещает об изменеии свойств. Оповещает об изменениях.
        /// </summary>
        protected void SetProperty([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

1.2 Models

1.2.1 LocalNotification.cs

```
using System;

namespace appUI.Models
{
    /// <summary>
    /// Класс для локальных уведомлений.
    /// </summary>
    public class LocalNotification
    {
        /// <summary>
        /// Заголовок уведомления.
        /// </summary>
        public string Title { get; set; }
        /// <summary>
        /// Текст уведомления.
        /// </summary>
        public string Body { get; set; }
        /// <summary>
        /// Идентификатор уведомления.
        /// </summary>
        public int Id { get; set; }
        /// <summary>
        /// Индекс иконки уведомления.
        /// </summary>
        public int IconId { get; set; }
        /// <summary>
        /// Дата и время уведомления.
        /// </summary>
        public DateTime NotifyTime { get; set; }
    }
}
```

1.2.2 TaskItem.cs

```
using SQLite;
using System;
using System.ComponentModel;
using Newtonsoft.Json;
using Xamarin.Forms;

namespace appUI.Models
{
    /// <summary>
    /// Модель задачи.
    /// </summary>
    [Serializable]
    public class TaskItem : INotifyPropertyChanged
    {
        /// <summary>
        /// Заголовок задания.
        /// </summary>
        string taskTitle;
        /// <summary>
        /// Флаг, показывающий выполнено или не выполнено задание.
        /// </summary>
        bool isDone;
        /// <summary>
        /// Идентификотор задания.
        /// </summary>
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
        /// <summary>
        /// Заголовок задания. Оповещает об изменениях.
        /// </summary>
        public string TaskTitle {
            get => taskTitle;
            set {
                if(value != taskTitle)
                    SetProperty(ref taskTitle, value, nameof(TaskTitle));
            }
        }
        /// <summary>
        /// Флаг, показывающий выполнено или не выполнено задание.
        /// Оповещает об изменениях.
        /// </summary>
        public bool IsDone { get => isDone;
            set {
                if(isDone != value)
                {
                    SetProperty(ref isDone, value, nameof(IsDone));
                    PropertyChanged?.Invoke(
```

```
                    this, new PropertyChangedEventArgs(nameof(MyColor)));
                PropertyChanged?.Invoke(
                    this, new PropertyChangedEventArgs(nameof(MyTextDecorations)));
            }
        }
    }
    /// <summary>
    /// Цвет текста.
    /// </summary>
    [Ignore, JsonIgnore]
    public Color MyColor { get =>IsDone ? Color.DarkGray : Color.DimGray; }
    /// <summary>
    /// Стиль текста: перечеркнутый или нормальный текст.
    /// </summary>
    [Ignore, JsonIgnore]
    public TextDecorations MyTextDecorations { get => IsDone ?
TextDecorations.Strikethrough : TextDecorations.None;}
    /// <summary>
    /// Свойство для оповещения об изменениях.
    /// </summary>
    public event PropertyChangedEventHandler PropertyChanged;
    /// <summary>
    /// Устанавливает значения и оповещает об изменениях.
    /// </summary>
    /// <typeparam name="T">Обобщение для всех типов.</typeparam>
    /// <param name="property">Ссылка на свойство.</param>
    /// <param name="value">Значение для установки.</param>
    /// <param name="name">Имя свойства.</param>
    protected void SetProperty<T>(ref T property, T value, string name)
    {
        property = value;
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
  }
}
```

1.2.3 TaskPageModel.cs

```
using Newtonsoft.Json;
using SQLite;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Xamarin.Forms;

namespace appUI.Models
{
    /// <summary>
    /// Модель страницы Modal Tasks.
    /// </summary>
    [Serializable]
    public class TaskPageModel : INotifyPropertyChanged
    {
        /// <summary>
        /// Свойство для оповещения об изменениях.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// Заголовок страницы.
        /// </summary>
        string pageTitle;
        /// <summary>
        /// Иконка страницы.
        /// </summary>
        string pageIcon;
        /// <summary>
        /// Дата выполнения.
        /// </summary>
        DateTime date;
        /// <summary>
        /// Время выполнения.
        /// </summary>
        TimeSpan time;
        /// <summary>
        /// Флаг важности страницы: важен или не важен.
        /// </summary>
        bool isImportant;
        /// <summary>
        /// Флаг уведомления. Включен или не включен.
        /// </summary>
        bool isRemind;
        /// <summary>
        /// Идентификатор уведомления.
        /// </summary>
        public int? NotificfId { get; set; }
```

```
/// <summary>
/// Идентификатор данной страницы.
/// </summary>
[PrimaryKey, AutoIncrement]
public int Id { get; set; }
/// <summary>
/// Заголовок страницы. Оповещает о изменениях.
/// </summary>
public string PageTitle
{
    get => pageTitle;
    set {
        pageTitle = value;
        if (value != pageTitle)
            PropertyChanged?.Invoke(
                this, new PropertyChangedEventArgs(nameof(PageTitle)));
    }
}
/// <summary>
/// Иконка страницы. Оповещает о изменениях.
/// </summary>
public string PageIcon
{
    get => pageIcon;
    set {
        if (value != pageIcon)
            SetProperty(ref pageIcon, value, nameof(PageIcon));
    }
}
/// <summary>
/// Горит звезда или нет.
/// </summary>
[Ignore, JsonIgnore]
public string StarIcon { get => IsImportant ? "star.png" : "nstar.png"; }
/// <summary>
/// Флаг важности страницы: важен или не важен.
/// Оповещает о изменениях.
/// </summary>
public bool IsImportant
{
    get => isImportant;
    set {
        if (value != isImportant)
        {
            SetProperty(ref isImportant, value, nameof(IsImportant));
            PropertyChanged?.Invoke(
                this, new PropertyChangedEventArgs(nameof(StarIcon)));
        }
    }
```

```
}
/// <summary>
/// Флаг уведомления. Включен или не включен.
/// Оповещает о изменениях.
/// </summary>
public bool IsRemind
{
    get => isRemind;
    set {
        if(value != isRemind)
        {
            SetProperty(ref isRemind, value, nameof(IsRemind));
            PropertyChanged?.Invoke(
                this, new PropertyChangedEventArgs(nameof(RemindIcon)));
        }
    }
}
/// <summary>
/// Включенный звонок или не включенный.
/// </summary>
[Ignore, JsonIgnore]
public string RemindIcon
{
    get => isRemind ? "bell.png" : "nbell.png";
}
/// <summary>
/// Выполнены ли все задания.
/// </summary>
public bool IsDone { get; set; }
/// <summary>
/// Цвет текста.
/// </summary>
[Ignore, JsonIgnore]
public Color Color
{
    get {
        if (IsDone)
            return Color.Green;
        else
            return Color.DimGray;

    }
}
/// <summary>
/// Дата выполнения. Оповещает об изменениях.
/// </summary>
public DateTime Date { get => date.Date;
    set {
        if(value != date)
```

```
                SetProperty(ref date, value, nameof(Date));
            }
        }
        /// <summary>
        /// Время выполнения. Оповещает об изменениях.
        /// </summary>
        public TimeSpan Time {
            get => time;
            set {
                if(value != time)
                    SetProperty(ref time, value, nameof(Time));
            }
        }
        /// <summary>
        /// Задачи страницы.
        /// </summary>
        [Ignore, JsonIgnore]
        public ObservableCollection<TaskItem> ToDoList { get; set; }
        /// <summary>
        /// Задачи страницы в виде строки(Json).
        /// </summary>
        public string ListToString { get; set; }
        /// <summary>
        /// Устанавливает значения и оповещает об изменениях.
        /// </summary>
        /// <typeparam name="T">Обобщение для всех типов.</typeparam>
        /// <param name="property">Ссылка на свойство.</param>
        /// <param name="value">Значение для установки.</param>
        /// <param name="name">Имя свойства.</param>
        protected void SetProperty<T>(ref T property, T value, string name)
        {
            property = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

1.3 pages

1.3.1 AuthPage.cs

```
using appUI.DataModels;
using appUI.Persistents;
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Collections.Generic;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace appUI.pages
{
    /// <summary>
    /// Страница для регистрации\авторизации пользователя.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class AuthPage : ContentPage
    {
        /// <summary>
        /// Интерфейс для взаимодействия с Firebase Authentication SDK.
        /// </summary>
        readonly IFirebaseAuth auth;
        /// <summary>
        /// Узел пользователей в базе данных Firebase Realtime Database.
        /// </summary>
        readonly string userPath = "Users";
        /// <summary>
        /// Интерфейс для оповещений.
        /// </summary>
        private readonly ILocalNotificationService notificationManager;
        /// <summary>
        /// Ссылка для доступа к базе данных.
        /// </summary>
        private static readonly string fbUrl = "https://vobshage-e86bd.firebaseio.com/";
        /// <summary>
        /// Предоставляет доступ к онлайн базе данных.
        /// </summary>
        readonly FirebaseClient firebase;
        /// <summary>
        /// Установка начальных значений
        /// </summary>
        public AuthPage()
        {
            InitializeComponent();
            auth = DependencyService.Get<IFirebaseAuth>();
```

```
            notificationManager = DependencyService.Get<ILocalNotificationService>();
            firebase  = new FirebaseClient(fbUrl);
        }
        /// <summary>
        /// Вход и регистрация в системе Firebase Authentication SDK.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Входные данные.</param>
        private async void LoginBtn_Clicked(object sender, EventArgs e)
        {
            try
            {
                string Token;
                if (LoginBtn.Text == "Login")
                {
                    Token = await auth.LoginWithEmailPassword(EmailInput.Text,
PasswordInput.Text);
                    if (Token != "")
                    {
                        Application.Current.MainPage = new MainPage();
                    }
                    else
                    {
                        notificationManager.LongAlert("E-mail or password are incorrect.Try
again!");
                    }
                }
                else
                {
                    Token = await auth.CreateWithEmailPassword(EmailInput.Text,
PasswordInput.Text);

                    if (Token != "")
                    {
                        var user = new User { Name = EmailInput.Text.Split('@')[0], Id =
auth.GetUserID(), TeamsId = new List<string>() };

                        user.Token = (await firebase.Child(userPath).PostAsync(user)).Key;

                        await firebase.Child(userPath).Child(user.Token).PutAsync(user);

                        Application.Current.MainPage = new MainPage();
                    }
                    else
                    {
                        notificationManager.LongAlert("E-mail or password are incorrect.
Password shoud be longer. Try again! ");
                    }
                }
```

```
            }
            catch
            {
                notificationManager.LongAlert("Problems with Internet connectivity. Please
try again.");
            }
        }
        /// <summary>
        /// Меняет функции авторизации\входа.
        /// </summary>
        /// <param name="sender">Отправель.</param>
        /// <param name="e">Входные данные.</param>
        private void NewAccountBtn_Clicked(object sender, EventArgs e)
        {
            var str = LoginBtn.Text;
            LoginBtn.Text = NewAccountBtn.Text;
            NewAccountBtn.Text = str;
        }
    }
}
```

1.3.2 CheckListPage.cs

```
using appUI.DataModels;
using appUI.IListeners;
using appUI.Models;
using appUI.Persistents;
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Collections.ObjectModel;
using System.Threading.Tasks;
using Xamarin.Essentials;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace appUI.pages
{
    /// <summary>
    /// Страницы командного чек-листа.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CheckListPage : ContentPage
    {
        /// <summary>
        /// Модель страницы.
        /// </summary>
        CheckList CurrentCheckList;
        /// <summary>
        /// Предоставляет доступ к онлайн базе данных.
        /// </summary>
        readonly FirebaseClient firebase;
        /// <summary>
        /// Интерфейс для оповещений.
        /// </summary>
        readonly ILocalNotificationService notification;
        /// <summary>
        /// Интерфейс для определения изменения данных.
        /// </summary>
        readonly IMyValueListener checListListener;
        /// <summary>
        /// URL онлайн базы данных.
        /// </summary>
        private static readonly string fbUrl =
            "https://vobshage-e86bd.firebaseio.com/";
        /// <summary>
        /// Название узла в онлайн базе данных Firbase
        /// Realtime database.
        /// </summary>
        readonly string checListsPath = "CheckLists";
```

```
/// <summary>
/// Задание начальных значений.
/// </summary>
/// <param name="checkList"></param>
public CheckListPage(CheckList checkList)
{
    InitializeComponent();
    firebase = new FirebaseClient(fbUrl);
    notification = DependencyService.Get<ILocalNotificationService>();
    checListListener = DependencyService.Get<IMyValueListener>();
    CurrentCheckList = checkList;

    MyDatePicker.MinimumDate = DateTime.Today;
    MyDatePicker.MinimumDate = DateTime.Today.AddSeconds(1);
    MyTimePicker.Time = new TimeSpan(9, 0, 1);
    BindingContext = CurrentCheckList;
}
/// <summary>
/// Подписка на события.
/// </summary>
protected override void OnAppearing()
{
    try
    {
        if (CurrentCheckList != null)
        {
            if (CurrentCheckList.Tasks == null)
                CurrentCheckList.Tasks = new ObservableCollection<TaskItem>();
            string path = checListsPath + "/" + CurrentCheckList.Id;

            checListListener.Subscribe(path);
            checListListener.OnChange += TasksList_Refreshing;

        }
        else
        {
            notification.LongAlert("Please reload check list");
        }
    }
    catch
    {
        notification.LongAlert("Please check internet connection.");
    }
    base.OnAppearing();
}
/// <summary>
/// Сохраняет привязанный чек-лист в базе данных.
/// </summary>
async void SaveCurrentCheckList()
```

```
{
    try
    {
        if (CurrentCheckList != null)
        {
            await firebase.Child(checListsPath)
                .Child(CurrentCheckList.Id).PutAsync(CurrentCheckList);
        }
    }
    catch
    {
        notification.LongAlert("Please check internet connection.");
    }
}
/// <summary>
/// Добавляет задачу в чек-лист.
/// </summary>
/// <param name="sender">Отправителью</param>
/// <param name="e">Данные.</param>
private async void PlusFab_Clicked(object sender, EventArgs e)
{
    try
    {
        var title = await Input("New task", "Enter your task here", "Task", "Add");
        if (title == null)
            return;
        var task = new TaskItem { TaskTitle = title };
        if (CurrentCheckList.Tasks == null)
            CurrentCheckList.Tasks = new ObservableCollection<TaskItem>();
        CurrentCheckList.Tasks.Add(task);
        SaveCurrentCheckList();
    }
    catch
    {
        notification.LongAlert("Please check internet connection.");
    }
}
/// <summary>
/// Метод для ввода строки.
/// </summary>
/// <param name="title">Заголовок.</param>
/// <param name="message">Сообщение.</param>
/// <param name="placeholder">Текст при пустом вводе.</param>
/// <param name="accept">Текст принятия действий.</param>
/// <param name="initialValue">Начальное значение.</param>
/// <returns>Полученная строка.</returns>
async Task<string> Input(string title, string message,
    string placeholder, string accept, string initialValue = "")
{
```

```
        var temp = await DisplayPromptAsync(title, message, placeholder: placeholder,
           accept: accept, initialValue: initialValue);
        if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
        {
            temp = temp.Trim(' ');
            if (temp.Length == 1)
                temp += " ";
            temp = char.ToUpper(temp[0]) + temp.Substring(1);
        }
        else if (temp != null)
        {
            await DisplayAlert("Empty input", "Please enter correct value", "Ok");
            temp = null;
        }

        return temp;
    }
    /// <summary>
    /// Метод срабатывает когда происходит нажатие на элемент списка.
    /// Нужно, чтобы цветом не выделялся выбранный элемент.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void TasksList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
    {
        TasksList.SelectedItem = null;
    }
    /// <summary>
    /// Метод срабатывает когда происходит нажатие на элемент списка.
    /// Отмечает, что задание выполнено \ не выполнено.
    /// </summary>
    /// <param name="sender">Отправитель.</param>
    /// <param name="e">Экземпляр с данными.</param>
    private void TasksList_ItemTapped(object sender, ItemTappedEventArgs e)
    {
        var item = (e.Item as TaskItem);
        item.IsDone = !item.IsDone;
        SaveCurrentCheckList();
    }
    /// <summary>
    /// Метод для редактирования текста задачи.
    /// </summary>
    /// <param name="sender">Отправитель.</param>
    /// <param name="e">Экземпляр с данными.</param>
    private async void EditTask_Clicked(object sender, EventArgs e)
    {
        try
        {
            var item = (TaskItem)((MenuItem)sender)?.BindingContext;
```

```
            if (item != null)
            {
                string temp = await Input("Edit the task", "Please enter here",
                    "Task", "Save", item.TaskTitle);
                if (temp != null)
                {
                    item.TaskTitle = temp;
                    SaveCurrentCheckList();
                }
            }
        }
        catch
        {
            notification.LongAlert("Please check internet connection.");
        }
    }
    /// <summary>
    /// Метод для удаления задачи из листа.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void DeleteTask_Clicked(object sender, EventArgs e)
    {
        var item = (TaskItem)((MenuItem)sender)?.BindingContext;
        if (item != null)
        {
            CurrentCheckList.Tasks.Remove(item);
            SaveCurrentCheckList();
        }
    }
    /// <summary>
    /// Обновление листа.
    /// </summary>
    /// <param name="sender">Отправитель.</param>
    /// <param name="e">Экземпляр с данными.</param>
    private async void TasksList_Refreshing(object sender, EventArgs e)
    {
        try
        {
            if (CurrentCheckList != null)
            {
                CurrentCheckList = await firebase.Child(checListsPath)
                    .Child(CurrentCheckList.Id).OnceSingleAsync<CheckList>();
                TasksList.ItemsSource = CurrentCheckList.Tasks;
                HeaderLbl.Text = CurrentCheckList.Title;
                TasksList.IsRefreshing = false;
            }
        }
        catch{
```

```
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Срабатывает после выбора даты.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private void MyDatePicker_DateSelected(object sender, DateChangedEventArgs e)
        {
            CurrentCheckList.Date = e.NewDate;
            SaveCurrentCheckList();
        }
        /// <summary>
        /// Срабатывает при закрытии окно выбора времени.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private void MyTimePicker_Unfocused(object sender, FocusEventArgs e)
        {
            var time = MyTimePicker.Time;
            if (time.Seconds == 0)
            {
                CurrentCheckList.Time = new TimeSpan(time.Hours, time.Minutes, 1);
                SaveCurrentCheckList();
            }
        }
        /// <summary>
        /// Копирование текста задачи.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private async void CopyId_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (TaskItem)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    await Clipboard.SetTextAsync(item.TaskTitle);
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
    }
}
```

1.3.3 GeneralPage.cs

```
using appUI.DataModels;
using appUI.Persistents;
using Firebase.Database;
using Firebase.Database.Query;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Threading.Tasks;
using Xamarin.Essentials;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace appUI.pages
{
    /// <summary>
    /// Страница для работы с командами.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class GeneralPage : ContentPage
    {
        /// <summary>
        /// URL онлайн базы данных.
        /// </summary>
        private static readonly string fbUrl = "https://vobshage-e86bd.firebaseio.com/";
        /// <summary>
        /// Предоставляет доступ к онлайн базе данных.
        /// </summary>
        readonly FirebaseClient firebase = new FirebaseClient(fbUrl);
        /// <summary>
        /// Интерфейс для работы с системой регистрации\авторизации.
        /// </summary>
        readonly IFirebaseAuth auth;
        /// <summary>
        /// Интерфейс для оповещения пользователя.
        /// </summary>
        private readonly ILocalNotificationService notification;
        /// <summary>
        /// Название узла команд в онлайн базе данных.
        /// </summary>
        const string teamPath = "Teams";
        /// <summary>
        /// Название узла пользователей в онлайн базе данных.
        /// </summary>
        const string usersPath = "Users";
        /// <summary>
        /// Текущий пользователь.
```

```
        /// </summary>
        User CurrentUser;
        /// <summary>
        /// Команды пользователя.
        /// </summary>
        ObservableCollection<Team> Teams;
        /// <summary>
        /// Задание начальных значений.
        /// </summary>
        public GeneralPage()
        {
            InitializeComponent();
            auth = DependencyService.Get<IFirebaseAuth>();
            notification = DependencyService.Get<ILocalNotificationService>();
        }
        /// <summary>
        /// Задание "тяжелых" начальных значений.
        /// </summary>
        protected async override void OnAppearing()
        {
            try
            {
                CurrentUser = (await firebase.Child(usersPath).OnceAsync<User>())
                        .Where(a => a.Object.Id == auth.GetUserID()).First().Object;
                if (CurrentUser.TeamsId == null)
                    CurrentUser.TeamsId = new List<string>();
                var teams = await GetTeamsAsync(CurrentUser.TeamsId);
                Teams = new ObservableCollection<Team>(teams);
            }
            catch { notification.LongAlert("Please check internet connection."); }

            if (Teams == null)
                Teams = new ObservableCollection<Team>();
            MainList.ItemsSource = Teams;

            base.OnAppearing();
        }
        /// <summary>
        /// Получение команд из онлайн базы данных.
        /// </summary>
        /// <param name="ids">Идентификаторы команд.</param>
        /// <returns></returns>
        async Task<Team[]> GetTeamsAsync(List<string> ids)
        {
            Team[] teams = new Team[ids.Count];
            for (int i = 0; i < ids.Count; i++)
            {
                teams[i] = await
  firebase.Child(teamPath).Child(ids[i]).OnceSingleAsync<Team>();
```

```
            }
            return teams;
        }
        /// <summary>
        /// Добавление\создание команд.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void PlusFab_Clicked(object sender, EventArgs e)
        {
            string act = await DisplayActionSheet("Select mode", "Cancel",
                null, "Join the team", "Create new team");
            if (act == "Join the team")
            {
                //Для присоединения к существующей команде.
                string id = await Input("Team ID", "Enter here secret team ID",
                    "Team ID", "Join");
                try
                {
                    if (id == null)
                        return;
                    var team = await firebase.Child(teamPath + "/" + id)
                        .OnceSingleAsync<Team>();

                    Teams.Add(team);

                    CurrentUser.TeamsId.Add(id);
                    await
firebase.Child(usersPath).Child(CurrentUser.Token).PutAsync(CurrentUser);
                }
                catch
                {
                    notification.LongAlert("Incorrect id. Please try again");
                }
            }
            else if (act == "Create new team")
            {
                //Создание новой команды.
                try
                {
                    string name = await Input("Team name", "Enter here team name", "Team
name", "Create new team");
                    if (name == null)
                        return;
                    var team = new Team
                    {
                        Name = name,
                        MembersToken = new List<string> { CurrentUser.Token }
                    };
```

```
                    team.Id = (await firebase.Child(teamPath).PostAsync(team)).Key;
                    await firebase.Child(teamPath).Child(team.Id).PutAsync(team);

                    Teams.Add(team);

                    CurrentUser.TeamsId.Add(team.Id);
                    await
firebase.Child(usersPath).Child(CurrentUser.Token).PutAsync(CurrentUser);
                }
                catch
                {
                    notification.LongAlert("Please try again");
                }
            }
        }
        /// <summary>
        /// Метод для ввода строки.
        /// </summary>
        /// <param name="title">Заголовок.</param>
        /// <param name="message">Сообщение.</param>
        /// <param name="placeholder">Текст при пустом вводе.</param>
        /// <param name="accept">Текст принятия действий.</param>
        /// <param name="initial">Начальное значение.</param>
        /// <returns>Полученная строка.</returns>
        async Task<string> Input(string title, string message, string placeholder, string
accept, string initial = "")
        {
            var temp = await DisplayPromptAsync(title, message, placeholder: placeholder,
               accept: accept, initialValue: initial);
            if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
            {
                temp = temp.Trim(' ');
                if (temp.Length == 1)
                    temp += " ";
                temp = char.ToUpper(temp[0]) + temp.Substring(1);
            }
            else if (temp != null)
            {
                await DisplayAlert("Empty input", "Please enter value", "Ok");
                temp = null;
            }

            return temp;
        }
        /// <summary>
        /// Выход из системы авторизации\регистрации.
        /// </summary>
        /// <param name="sender">Отправтель</param>
        /// <param name="e">Экземпляр с данными.</param>
```

```
private void LogOut_Clicked(object sender, EventArgs e)
{
    try
    {
        if (Connectivity.NetworkAccess == NetworkAccess.Internet)
        {
            auth.SignOut();
            Application.Current.MainPage = new AuthPage();
        }
        else
        {
            notification.LongAlert("Please check internet connection.");
        }
    }
    catch
    {
        notification.LongAlert("Please check internet connection.");
    }
}
/// <summary>
/// Метод срабатывает когда происходит нажатие на элемент списка.
/// Нужно, чтобы цветом не выделялся выбранный элемент.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainList_ItemSelected(object sender, SelectedItemChangedEventArgs e) =>
    MainList.SelectedItem = null;
/// <summary>
/// Метод срабатывает когда происходит нажатие на элемент списка.
/// Отмечает, что задание выполнено \ не выполнено.
/// </summary>
/// <param name="sender">Отправитель.</param>
/// <param name="e">Экземпляр с данными.</param>
private async void MainList_ItemTapped(object sender, ItemTappedEventArgs e)
{
    if (Connectivity.NetworkAccess == NetworkAccess.Internet)
    {
        var item = e.Item as Team;
        await Navigation.PushAsync(new TeamsPage(CurrentUser, item));
    }
    else
    {
        notification.LongAlert("Please check internet connection.");
    }
}
/// <summary>
/// Метод для редактирования названия команды.
/// </summary>
/// <param name="sender">Отправитель.</param>
```

```
        /// <param name="e">Экземпляр с данными.</param>
        private async void TeamEdit_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (Team)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    string temp = await Input("Edit team's name", "Please enter here",
                        "Team name", "Save", item.Name);
                    if (temp != null)
                    {
                        item.Name = temp;
                        await firebase.Child(teamPath).Child(item.Id).PutAsync(item);
                    }
                }
            }
            catch
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Метод для удаления команды.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private async void TeamDelete_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (Team)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    CurrentUser.TeamsId.Remove(item.Id);
                    Teams.Remove(item);
                    await
firebase.Child(usersPath).Child(CurrentUser.Token).PutAsync(CurrentUser);

                }
            }
            catch
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Копирование идентификатора команды.
        /// </summary>
```

```
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private async void CopyId_Clicked(object sender, EventArgs e)
        {
            var item = (Team)((MenuItem)sender)?.BindingContext;
            if (item != null)
            {
                await Clipboard.SetTextAsync(item.Id);
            }
        }
        /// <summary>
        /// Показ окна с подсказками.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        private void HelpTb_Clicked(object sender, EventArgs e)
        {
            Navigation.PushAsync(new HelpPage());
        }
    }
}
```

1.3.4 HelpPage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace appUI.pages
{
    /// <summary>
    /// Страница с подсказками.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class HelpPage : ContentPage
    {
        /// <summary>
        /// Установка начальных значений.
        /// </summary>
        public HelpPage()
        {
            InitializeComponent();
        }
    }
}
```

1.3.5 ModalTasks.cs

```
using appUI.Models;
using appUI.Persistents;
using SQLite;
using System;
using System.Collections.ObjectModel;
using System.Text.Json;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using System.Linq;
using Xamarin.Essentials;

namespace appUI.pages
{
    /// <summary>
    /// Страница персонального чек-листа.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ModalTasks : ContentPage
    {
        /// <summary>
        /// Предостовляет асинхронное соединение с локальной базой данных.
        /// </summary>
        readonly SQLiteAsyncConnection connection;
        /// <summary>
        /// Интерфейс для оповещения пользователя.
        /// </summary>
        private readonly ILocalNotificationService notification;
        /// <summary>
        /// Модель страницы.
        /// </summary>
        public TaskPageModel Model { get; private set; }
        /// <summary>
        /// Задание начальных значений.
        /// </summary>
        /// <param name="pageModel"></param>
        /// <param name="connection"></param>
        public ModalTasks(TaskPageModel pageModel, SQLiteAsyncConnection connection)
        {
            InitializeComponent();
            this.connection = connection;
            Model = pageModel;
            MyDatePicker.MinimumDate = DateTime.Today.AddSeconds(1);
            MyTimePicker.Time = new TimeSpan(9, 0, 1);
            BindingContext = Model;
            notification = DependencyService.Get<ILocalNotificationService>();

        }
```

```
        /// <summary>
        /// Задание "тяжелых" начальных значений.
        /// </summary>
        protected override void OnAppearing()
        {
            if (Model.ListToString != null)
            {
                try
                {
                    Model.ToDoList =
JsonSerializer.Deserialize<ObservableCollection<TaskItem>>(Model.ListToString);
                }
                catch
                {
                    notification.LongAlert("Data is corrupted, please add new tasks.");
                    Model.ToDoList = new ObservableCollection<TaskItem>();
                }
            }
            else
            {
                Model.ToDoList = new ObservableCollection<TaskItem>();
            }
            TasksList.ItemsSource = Model.ToDoList;
            base.OnAppearing();
        }
        /// <summary>
        /// Отметка\снятие галочки о выполнении.
        /// </summary>
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void TasksList_ItemTapped(object sender, ItemTappedEventArgs e)
        {
            try
            {
                var item = (e.Item as TaskItem);
                item.IsDone = !item.IsDone;
                Model.ListToString = JsonSerializer.Serialize(Model.ToDoList);
                if (!Model.ToDoList.Any(el => el.IsDone == false))
                    Model.IsDone = true;
                else
                    Model.IsDone = false;
                await connection.UpdateAsync(Model);
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
```

```
        /// Метод для редактирования текста задачи.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Экземпляр с данными.</param>
        async private void Edit_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (TaskItem)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    var temp = await DisplayPromptAsync("Complete", "edit your notes",
initialValue: $"{item.TaskTitle}");
                    if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
                    {
                        temp = temp.Trim(' ');
                        item.TaskTitle = temp;
                        Model.ListToString = JsonSerializer.Serialize(Model.ToDoList);
                        await connection.UpdateAsync(Model);
                    }
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }

        }
        /// <summary>
        /// Метод для удаления задачи из листа.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void Delete_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (TaskItem)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    Model.ToDoList.Remove(item);
                    Model.ListToString = JsonSerializer.Serialize(Model.ToDoList);
                    await connection.UpdateAsync(Model);
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
```

```
        }
        /// <summary>
        /// Добавляет задачу в чек-лист.
        /// </summary>
        /// <param name="sender">Отправителю</param>
        /// <param name="e">Данные.</param>
        async private void Fabplus_Clicked(object sender, EventArgs e)
        {
            try
            {
                var temp = await DisplayPromptAsync("New task", "Enter task title", accept:
"Create task",
                    placeholder: "Title");
                if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
                {
                    temp = temp.Trim(' ');
                    Model.ToDoList.Add(new TaskItem { TaskTitle = temp });
                    TasksList.ScrollTo(Model.ToDoList[Model.ToDoList.Count - 1],
ScrollToPosition.End, true);
                    Model.ListToString = JsonSerializer.Serialize(Model.ToDoList);
                    await connection.UpdateAsync(Model);

                }
                else if (temp != null)
                    notification.LongAlert("Empty title. Please try enter correct title");
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
        /// Включение\отключение уведомлений.
        /// </summary>
        /// <param name="sender">Отправителю</param>
        /// <param name="e">Данные.</param>
        private async void RemaindTb_Clicked(object sender, EventArgs e)
        {
            try
            {
                Model.IsRemind = !Model.IsRemind;
                if (Model.IsRemind)
                {
                    var date = new DateTime(Model.Date.Year, Model.Date.Month,
Model.Date.Day, Model.Time.Hours, Model.Time.Minutes, 0);
                    if (date < DateTime.Now)
                    {
                        Model.IsRemind = false;
                        notification.LongAlert("Please select correct time");
```

```
                    return;
                }
                if (Model.NotificfId == null)
                    Model.NotificfId = Guid.NewGuid().GetHashCode();
                MyDatePicker.IsEnabled = false;
                MyTimePicker.IsEnabled = false;
                notification.LocalNotification("Remainder", Model.PageTitle,
(int)Model.NotificfId, date);
                notification.ShortAlert("Remind you at " + date.ToString("hh:mm tt, MMM
dd"));
            }
            else
            {
                MyDatePicker.MinimumDate = DateTime.Today.AddSeconds(1);
                MyTimePicker.Time = new TimeSpan(9, 0, 1);
                notification.Cancel((int)Model.NotificfId);
                MyDatePicker.IsEnabled = true;
                MyTimePicker.IsEnabled = true;

            }
            await connection.UpdateAsync(Model);
        }
        catch
        {
            notification.LongAlert("Please try again.");
        }
    }
    /// <summary>
    /// Устанавливает приоритетность чек-листа.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private async void StarTb_Clicked(object sender, EventArgs e)
    {
        try
        {
            Model.IsImportant = !Model.IsImportant;
            await connection.UpdateAsync(Model);
        }
        catch
        {
            notification.LongAlert("Please try again.");
        }
    }
    /// <summary>
    /// Метод срабатывает когда происходит нажатие на элемент списка.
    /// Нужно, чтобы цветом не выделялся выбранный элемент.
    /// </summary>
    /// <param name="sender"></param>
```

```
        /// <param name="e"></param>
        private void TasksList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
            => TasksList.SelectedItem = null;
        /// <summary>
        /// Копирует текст задания.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private async void CopyId_Clicked(object sender, EventArgs e)
        {
            var item = (TaskItem)((MenuItem)sender)?.BindingContext;
            if (item != null)
            {
                await Clipboard.SetTextAsync(item.TaskTitle);
            }
        }
    }
}
```

1.3.6 PersonalPage.cs

```
using appUI.Models;
using appUI.Persistents;
using SQLite;
using System;
using System.Collections.ObjectModel;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace appUI.pages
{
    /// <summary>
    /// Сраница с персональными чек-листами.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PersonalPage : ContentPage
    {
        /// <summary>
        /// Предостовляет асинхронное соединение с локальной базой данных.
        /// </summary>
        readonly SQLiteAsyncConnection sqConnection;
        /// <summary>
        /// Интерфейс для работы с системой регистрации\авторизации.
        /// </summary>
        readonly IFirebaseAuth auth;
        /// <summary>
        /// Интерфейс для оповещения пользователя.
        /// </summary>
        readonly ILocalNotificationService notification;
        /// <summary>
        /// Модели чек-листов.
        /// </summary>
        public ObservableCollection<TaskPageModel> Models { get; private set; }
        /// <summary>
        /// Задание начальных значений.
        /// </summary>
        public PersonalPage()
        {
            InitializeComponent();
            auth = DependencyService.Get<IFirebaseAuth>();
            sqConnection = DependencyService.Get<ISQLite>().
                GetConnectionPersonal(auth.GetUserID());
            if (Models == null)
                Models = new ObservableCollection<TaskPageModel>();
            notification = DependencyService.Get<ILocalNotificationService>();
        }
        /// <summary>
```

```
        /// Задание "тяжелых" начальных значений.
        /// </summary>
        protected async override void OnAppearing()
        {
            try
            {
                await sqConnection.CreateTableAsync<TaskPageModel>();
                await sqConnection.CreateTableAsync<TaskItem>();
                Models = new ObservableCollection<TaskPageModel>(
                    await sqConnection.Table<TaskPageModel>().ToArrayAsync());
                if (Models == null)
                    Models = new ObservableCollection<TaskPageModel>();
                MainList.ItemsSource = Models;
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
            base.OnAppearing();
        }
        /// <summary>
        /// Добавление персонального чек-листа.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        async void PlusFab_Clicked(object sender, EventArgs e)
        {
            try
            {
                var temp = await DisplayPromptAsync("New title", "Enter list's title",
placeholder: "Enter here",
                accept: "Create new list");
                if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
                {
                    temp = temp.Trim(' ');
                    temp = char.ToUpper(temp[0]) + temp.Substring(1);
                    var obj = new TaskPageModel { PageTitle = temp, Date = DateTime.Today };
                    await sqConnection.InsertAsync(obj);
                    Models.Add(obj);
                }
                else if (temp != null)
                    await DisplayAlert("Empty title", "Try enter correct title", "Ok");
                All_Clicked(null, null);
                if (Models.Count > 0)
                    MainList.ScrollTo(Models[Models.Count - 1], ScrollToPosition.End, true);
            }
            catch
            {
                notification.LongAlert("Please try again.");
```

```
            }
        }
        /// <summary>
        /// Показ страницы с чек-листом.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        async private void MainList_ItemTapped(object sender, ItemTappedEventArgs e) =>
            await Navigation.PushAsync(new ModalTasks((TaskPageModel)e.Item, sqConnection),
false);
        /// <summary>
        /// Устанавливает приоритетность чек-листу.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Important_Clicked(object sender, EventArgs e)
        {
            var temp = (TaskPageModel)((MenuItem)sender)?.BindingContext;
            temp.IsImportant = !temp.IsImportant;
        }
        /// <summary>
        /// Удаление чек-листа.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        async private void Delete_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (TaskPageModel)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    await sqConnection.DeleteAsync(item);
                    Models.Remove(item);
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
        /// Редактирование названия чек-листа.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        async private void Edit_Clicked(object sender, EventArgs e)
        {
            try
```

```
            {
                var item = (TaskPageModel)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    var temp = await DisplayPromptAsync("Complete", "Edit your list title",
initialValue: $"{item.PageTitle}");
                    if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
                    {
                        temp = temp.Trim(' ');
                        item.PageTitle = char.ToUpper(temp[0]) + temp.Substring(1);
                        await sqConnection.UpdateAsync(item);
                    }
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
        /// Показывает сегодняшние чек-листы.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        async private void MyDay_Clicked(object sender, EventArgs e)
        {
            try
            {
                var result = await Task.Run(MyDaySort);
                MainList.ItemsSource = result;
                ObservableCollection<TaskPageModel> MyDaySort()
                {
                    var temp = new ObservableCollection<TaskPageModel>();
                    foreach (var item in Models)
                        if (item.Date == DateTime.Today)
                        {
                            temp.Add(item);
                        }
                    return temp;
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
        /// Покзывает приоритетные чек-листы.
        /// </summary>
```

```
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>/// <param name="sender"></param>
        async private void ImportantLists_Clicked(object sender, EventArgs e)
        {
            try
            {
                var result = await Task.Run(MyDaySort);
                MainList.ItemsSource = result;
                ObservableCollection<TaskPageModel> MyDaySort()
                {
                    var temp = new ObservableCollection<TaskPageModel>();
                    foreach (var item in Models)
                        if (item.IsImportant)
                            temp.Add(item);
                    return temp;
                }
            }
            catch
            {
                notification.LongAlert("Please try again.");
            }
        }
        /// <summary>
        /// Показывает все чек-листы.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        private void All_Clicked(object sender, EventArgs e)
            => MainList.ItemsSource = Models;
        /// <summary>
        /// Метод срабатывает когда происходит нажатие на элемент списка.
        /// Нужно, чтобы цветом не выделялся выбранный элемент.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        private void MainList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
            => MainList.SelectedItem = null;
        /// <summary>
        /// Показывает страницу с подсказками.
        /// </summary>
        /// <param name="sender">Отправитель.</param>
        /// <param name="e">Данные.</param>
        private void HelpTb_Clicked(object sender, EventArgs e)
        {
            Navigation.PushAsync(new HelpPage());
        }
    }
}
```

1.3.7 TeamsPage.cs

```
using appUI.DataModels;
using appUI.Persistents;
using Firebase.Database;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

using appUI.IListeners;
using System.Collections.ObjectModel;
using Firebase.Database.Query;
using Xamarin.Essentials;

namespace appUI.pages
{
    /// <summary>
    /// Страница с чек-листами.
    /// </summary>
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class TeamsPage : ContentPage
    {
        /// <summary>
        /// Текущий пользователь.
        /// </summary>
        readonly User CurrentUser;
        /// <summary>
        /// Текущая команда.
        /// </summary>
        Team CurrentTeam;
        /// <summary>
        /// Интерфейс для работы с системой регистрации\авторизации.
        /// </summary>
        readonly FirebaseClient firebase;
        /// <summary>
        /// Интерфейс для оповещения пользователя.
        /// </summary>
        readonly ILocalNotificationService notification;
        /// <summary>
        /// Интерфейс для определения изменения данных.
        /// </summary>
        readonly IMyValueListener checListsListener;
        /// <summary>
        /// URL онлайн базы данных.
        /// </summary>
        private static readonly string fbUrl =
```

```
    "https://vobshage-e86bd.firebaseio.com/";
/// <summary>
/// Название узла чек-листов в онлайн базе данных Firbase
/// Realtime database.
/// </summary>
readonly string checListsPath = "CheckLists";
/// <summary>
/// Название узла команд в онлайн базе данных Firbase
/// Realtime database.
/// </summary>
readonly string teamsPath = "Teams";
/// <summary>
/// Чек-листы команды.
/// </summary>
public ObservableCollection<CheckList> CheckLists { get; set; }
/// <summary>
/// Задает начальные значения.
/// </summary>
/// <param name="user">Текущий пользователь.</param>
/// <param name="team">Текущая команда.</param>
public TeamsPage(User user, Team team)
{
    InitializeComponent();
    firebase = new FirebaseClient(fbUrl);
    notification = DependencyService.Get<ILocalNotificationService>();
    checListsListener = DependencyService.Get<IMyValueListener>();
    CurrentUser = user;
    CurrentTeam = team;
    Title = "Checklists";
    HeaderLabel.Text = CurrentTeam.Name;

}
/// <summary>
/// Задание "тяжелых" начальных значений.
/// </summary>
protected async override void OnAppearing()
{
    try
    {
        if (CurrentTeam != null && CurrentUser != null)
        {
            CheckLists = new ObservableCollection<CheckList>(await GetListsAsync());
            MainList.ItemsSource = CheckLists;
            string path = teamsPath + "/" + CurrentTeam.Id;
            checListsListener.Subscribe(path);
            checListsListener.OnChange += async (s, e) =>
            {
                CurrentTeam = await firebase.Child(path).OnceSingleAsync<Team>();
```

```
                    CheckLists = new ObservableCollection<CheckList>(await
GetListsAsync());
                    MainList.ItemsSource = CheckLists;
                };
            }
        }
        catch
        {
            CheckLists = new ObservableCollection<CheckList>();
            notification.LongAlert("Please check internet connection.");
        }
        base.OnAppearing();
    }
    /// <summary>
    /// Получение чек-листов из онлайн базы данных по их идентификаторам.
    /// </summary>
    /// <returns>Чек-листы.</returns>
    async Task<CheckList[]> GetListsAsync()
    {
        if (CurrentTeam.CheckListsId != null)
        {
            CheckList[] checkLists = new CheckList[CurrentTeam.CheckListsId.Count];
            for (int i = 0; i < CurrentTeam.CheckListsId.Count; i++)
            {
                checkLists[i] = await firebase.Child(checListsPath).
                    Child(CurrentTeam.CheckListsId[i]).OnceSingleAsync<CheckList>();
            }
            return checkLists;
        }
        else
            return new CheckList[0];
    }
    /// <summary>
    /// Добавление нового чек-листа.
    /// </summary>
    /// <param name="sender">Отправитель</param>
    /// <param name="e">Данные</param>
    private async void PlusFab_Clicked(object sender, EventArgs e)
    {
        try
        {
            string title = await Input("New check list title", "Enter your title",
"Title", "Create new check list");
            if (title == null)
                return;
            var checkList = new CheckList { Title = title };
            checkList.Id = (await
firebase.Child(checListsPath).PostAsync(checkList)).Key;
            await firebase.Child(checListsPath).Child(checkList.Id).PutAsync(checkList);
```

```
            if (CurrentTeam.CheckListsId == null)
                CurrentTeam.CheckListsId = new List<string>();
            CurrentTeam.CheckListsId.Add(checkList.Id);
            CheckLists.Add(checkList);

            await firebase.Child(teamsPath).Child(CurrentTeam.Id).PutAsync(CurrentTeam);
        }
        catch
        {
            notification.LongAlert("Please check your internet connection.");
        }

    }
    /// <summary>
    /// Метод для ввода строки.
    /// </summary>
    /// <param name="title">Заголовок.</param>
    /// <param name="message">Сообщение.</param>
    /// <param name="placeholder">Текст при пустом вводе.</param>
    /// <param name="accept">Текст принятия действий.</param>
    /// <param name="initialValue">Начальное значение.</param>
    /// <returns>Полученная строка.</returns>
    async Task<string> Input(string title, string message, string placeholder,
        string accept, string initialValue = "")
    {
        var temp = await DisplayPromptAsync(title, message, placeholder: placeholder,
           accept: accept, initialValue: initialValue);
        if (!string.IsNullOrEmpty(temp) && !string.IsNullOrWhiteSpace(temp))
        {
            temp = temp.Trim(' ');
            if (temp.Length == 1)
                temp += " ";
            temp = char.ToUpper(temp[0]) + temp.Substring(1);
        }
        else if (temp != null)
        {
            await DisplayAlert("Empty input", "Please enter correct value", "Ok");
            temp = null;
        }

        return temp;
    }
    /// <summary>
    /// Метод срабатывает когда происходит нажатие на элемент списка.
    /// Нужно, чтобы цветом не выделялся выбранный элемент.
    /// </summary>
    /// <param name="sender">Отправитель</param>
    /// <param name="e">Данные</param>
    private void MainList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
```

```
        {
            MainList.SelectedItem = null;
        }
        /// <summary>
        /// Показ страницы с чек-листом.
        /// </summary>
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void MainList_ItemTapped(object sender, ItemTappedEventArgs e)
        {
            if (Connectivity.NetworkAccess == NetworkAccess.Internet)
            {
                var item = e.Item as CheckList;
                await Navigation.PushAsync(new CheckListPage(item));
            }
            else
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Редактирование названия чек-листа.
        /// </summary>
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void EditCheckList_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (CheckList)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    string temp = await Input("Edit the check list", "Please enter here",
"Check list title", "Save", item.Title);
                    if (temp != null)
                    {
                        item.Title = temp;
                        await firebase.Child(checListsPath).Child(item.Id).PutAsync(item);
                    }
                }
            }
            catch
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Удаление чек-листа.
        /// </summary>
```

```
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void DeleteCheckList_Clicked(object sender, EventArgs e)
        {
            try
            {
                var item = (CheckList)((MenuItem)sender)?.BindingContext;
                if (item != null)
                {
                    CheckLists.Remove(item);
                    CurrentTeam.CheckListsId.Remove(item.Id);
                    await
firebase.Child(teamsPath).Child(CurrentTeam.Id).PutAsync(CurrentTeam);
                }
            }
            catch
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Обновление чек-листов.
        /// </summary>
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void MainList_Refreshing(object sender, EventArgs e)
        {
            try
            {
                if (CheckLists != null)
                {
                    CheckLists = new ObservableCollection<CheckList>(await GetListsAsync());
                    MainList.ItemsSource = CheckLists;
                    Title = CurrentTeam.Name;
                    MainList.IsRefreshing = false;
                }
            }
            catch
            {
                notification.LongAlert("Please check internet connection.");
            }
        }
        /// <summary>
        /// Копирование идентификтора команды.
        /// </summary>
        /// <param name="sender">Отправитель</param>
        /// <param name="e">Данные</param>
        private async void CopyId_Clicked(object sender, EventArgs e)
        {
```

```
            if (CurrentTeam != null)
            {
                await Clipboard.SetTextAsync(CurrentTeam.Id);
            }
        }
    }
}
```

2 Проект appUI.Anroid

2.1 Listeners

2.1.1 AppDataHelper.cs

```
using Android.App;
using Firebase;
using Firebase.Database;

namespace appUI.Droid.Listeners
{
    /// <summary>
    /// Класс для работы с базой данных Firebase Realtime Database.
    /// </summary>
    public static class AppDataHelper
    {
        /// <summary>
        /// Возвращает экземпляр для работы с базой данных Firebase Realtime Database.
        /// </summary>
        /// <returns></returns>
        public static FirebaseDatabase GetDatabase()
        {

            var app = FirebaseApp.InitializeApp(Application.Context);
            FirebaseDatabase database;

            if (app == null)
            {
                var option = new Firebase.FirebaseOptions.Builder()
                    .SetApplicationId("vobshage-e86bd")
                    .SetApiKey("AIzaSyDGPBtwHsbTwZxZfp-CmA-7wX3YUUA6jgs")
                    .SetDatabaseUrl("https://vobshage-e86bd.firebaseio.com")
                    .SetStorageBucket("vobshage-e86bd.appspot.com")
                    .Build();

                app = FirebaseApp.InitializeApp(Application.Context, option);
                database = FirebaseDatabase.GetInstance(app);
            }
            else
            {
                database = FirebaseDatabase.GetInstance(app);
            }

            return database;
        }
    }
}
```

2.1.2 CheckListListener.cs

```
using appUI.Droid.Listeners;
using appUI.Droid.Persistents;
using appUI.IListeners;
using appUI.Listeners;
using Firebase.Database;
using System;
using Xamarin.Forms;

[assembly: Dependency(typeof(CheckListListener))]
namespace appUI.Listeners
{
    /// <summary>
    /// Класс для получения изменений на подписанный узел.
    /// </summary>
    public class CheckListListener : Java.Lang.Object, IValueEventListener, IMyValueListener
    {
        /// <summary>
        /// Событие для получения уведомлений об измении в определенном узле базы данных.
        /// </summary>
        public event EventHandler OnChange;
        public void OnCancelled(DatabaseError error)
        {

        }
        /// <summary>
        /// Получение оповещения об изменении данных.
        /// </summary>
        /// <param name="snapshot">Обновленные данные.</param>
        public void OnDataChange(DataSnapshot snapshot)
        {
            OnChange?.Invoke(this, new EventArgs());
        }
        /// <summary>
        /// Устанавливает данный объект к узлу в базе данных, чтобы он оповещал о
изменениях.
        /// </summary>
        /// <param name="path"></param>
        public void Subscribe(string path)
        {
            try
            {
                DatabaseReference teamsRef = AppDataHelper.GetDatabase().GetReference(path);
                teamsRef.AddValueEventListener(this);
            }
            catch {
                var notification = new LocalNotificationService();
                notification.LongAlert("Please check your internet connection.");
```

```
            }
        }
    }
}
```

2.2 Persistents

2.2.1 LocalNotificationService.cs

```
using System;
using System.IO;
using System.Xml.Serialization;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Support.V4.App;
using Android.Widget;
using appUI.Droid.Persistents;
using appUI.Models;
using appUI.Persistents;
using Java.Lang;
using AndroidApp = Android.App.Application;

[assembly: Xamarin.Forms.Dependency(typeof(LocalNotificationService))]

namespace appUI.Droid.Persistents
{
    /// <summary>
    /// Класс для оповещения пользователя.
    /// </summary>
    public class LocalNotificationService : ILocalNotificationService
    {
        /// <summary>
        /// Unix-время
        /// </summary>
        readonly DateTime _jan1st1970 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
        /// <summary>
        /// Для задания уникального идентификатора.
        /// </summary>
        internal string _randomNumber;
        /// <summary>
        /// Установка локальных уведомлений на устройстве.
        /// </summary>
        /// <param name="title">Заголовокю</param>
        /// <param name="body">Текст сообщения.</param>
        /// <param name="id">Идентификатор уведомления.</param>
        /// <param name="notifyTime">Время уведомлениюю</param>
        /// <param name="repeateMs">Интервал повтора.</param>
        public void LocalNotification(string title, string body, int id, DateTime
notifyTime, long repeateMs = 0)
        {
            long totalMilliSeconds = (long)(notifyTime.ToUniversalTime() -
_jan1st1970).TotalMilliseconds;
            if (totalMilliSeconds < JavaSystem.CurrentTimeMillis())
```

```
            {
                totalMilliSeconds += repeateMs;
            }

            var intent = CreateIntent(id);
            var localNotification = new LocalNotification
            {
                Title = title,
                Body = body,
                Id = id,
                NotifyTime = notifyTime,
                IconId = Resource.Drawable.tick
            };

            var serializedNotification = SerializeNotification(localNotification);
            intent.PutExtra(ScheduledAlarmHandler.LocalNotificationKey,
serializedNotification);

            Random generator = new Random();
            _randomNumber = generator.Next(100000, 999999).ToString("D6");

            var pendingIntent = PendingIntent.GetBroadcast(Application.Context,
Convert.ToInt32(_randomNumber), intent, PendingIntentFlags.Immutable);
            var alarmManager = GetAlarmManager();
            if (repeateMs == 0)
                alarmManager.Set(AlarmType.RtcWakeup, totalMilliSeconds, pendingIntent);
            else
                alarmManager.SetRepeating(AlarmType.RtcWakeup, totalMilliSeconds, repeateMs,
pendingIntent);
        }
        /// <summary>
        /// Отмена уведомления.
        /// </summary>
        /// <param name="id">Идентификатор уведомления.</param>
        public void Cancel(int id)
        {
            var intent = CreateIntent(id);
            var pendingIntent = PendingIntent.GetBroadcast(Application.Context,
Convert.ToInt32(_randomNumber), intent, PendingIntentFlags.Immutable);
            var alarmManager = GetAlarmManager();
            alarmManager.Cancel(pendingIntent);
            var notificationManager = NotificationManagerCompat.From(Application.Context);
            notificationManager.CancelAll();
            notificationManager.Cancel(id);
        }
        /// <summary>
        /// Возвращает "хороший" механизм для выполнения задач в переднем плане.
        /// </summary>
        public static Intent GetLauncherActivity() =>
```

```
Application.Context.PackageManager.GetLaunchIntentForPackage(Application.Context.PackageName
);
        /// <summary>
        /// Возвращает  подзадачу, используется для показа уведомлений.
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private Intent CreateIntent(int id) => new Intent(Application.Context,
typeof(ScheduledAlarmHandler))
                .SetAction("LocalNotifierIntent" + id);
        /// <summary>
        /// Возвращает системный сервис для управления сигналами.
        /// </summary>
        private AlarmManager GetAlarmManager() =>
Application.Context.GetSystemService(Context.AlarmService) as AlarmManager;
        /// <summary>
        /// Сериализует уведомление.
        /// </summary>
        /// <param name="notification">Уведомление.</param>
        /// <returns>Сериализованное уведомление.</returns>
        private string SerializeNotification(LocalNotification notification)
        {
            var xmlSerializer = new XmlSerializer(notification.GetType());

            using (var stringWriter = new StringWriter())
            {
                xmlSerializer.Serialize(stringWriter, notification);
                return stringWriter.ToString();
            }
        }
        /// <summary>
        /// Показ длительнго Toast уведомления.
        /// </summary>
        /// <param name="message">Сообщение для показа</param>
        public void LongAlert(string message) => Toast.MakeText(Application.Context,
message, ToastLength.Long).Show();
        /// <summary>
        /// Показ кратковременного Toast уведомления.
        /// </summary>
        /// <param name="message">Сообщение для показа</param>
        public void ShortAlert(string message) => Toast.MakeText(Application.Context,
message, ToastLength.Short).Show();
    }
    /// <summary>
    /// Класс принимает уведомления.
    /// </summary>
    [BroadcastReceiver(Enabled = true, Label = "Local Notifications Broadcast Receiver")]
    public class ScheduledAlarmHandler : BroadcastReceiver
```

```
    {
        /// <summary>
        /// Уведомление пользователя о событии.
        /// </summary>
        NotificationManager manager;
        /// <summary>
        /// Ключ-константа.
        /// </summary>
        public const string LocalNotificationKey = "LocalNotification";
        /// <summary>
        /// Идентификатор канала.
        /// </summary>
        const string channelId = "default";
        /// <summary>
        /// Имя канала.
        /// </summary>
        const string channelName = "Default";
        /// <summary>
        /// Описание канала.
        /// </summary>
        const string channelDescription = "The default channel for notifications.";
        /// <summary>
        /// Флаг инициализации канала.
        /// </summary>
        bool channelInitialized = false;
        /// <summary>
        /// Этот метод вызывается, когда ScheduledAlarmHandler получает механизм для
вещания.
        /// </summary>
        /// <param name="context">Context в котором работает получатель.</param>
        /// <param name="intent">Полученный механизм(Intent)</param>
        public override void OnReceive(Context context, Intent intent)
        {
            if (!channelInitialized)
            {
                CreateNotificationChannel();
            }
            var extra = intent.GetStringExtra(LocalNotificationKey);
            var notification = DeserializeNotification(extra);
            //Generating notification
            var builder = new NotificationCompat.Builder(Application.Context, channelId)
                .SetContentTitle(notification.Title)
                .SetContentText(notification.Body)
                .SetSmallIcon(Resource.Drawable.tick)
                .SetDefaults((int)NotificationDefaults.Sound |
(int)NotificationDefaults.Vibrate);

            var resultIntent = LocalNotificationService.GetLauncherActivity();
            resultIntent.SetFlags(ActivityFlags.NewTask | ActivityFlags.ClearTask);
```

```
            var stackBuilder =
Android.Support.V4.App.TaskStackBuilder.Create(Application.Context);
            stackBuilder.AddNextIntent(resultIntent);

            Random random = new Random();
            int randomNumber = random.Next(9999 - 1000) + 1000;

            var resultPendingIntent =
                stackBuilder.GetPendingIntent(randomNumber,
(int)PendingIntentFlags.Immutable);
            builder.SetContentIntent(resultPendingIntent);
            // Sending notification
            manager.Notify(randomNumber, builder.Build());
        }
        /// <summary>
        /// Создание канала для оповещения.
        /// </summary>
        void CreateNotificationChannel()
        {
            manager =
(NotificationManager)AndroidApp.Context.GetSystemService(AndroidApp.NotificationService);

            if (Build.VERSION.SdkInt >= BuildVersionCodes.O)
            {
                var channelNameJava = new Java.Lang.String(channelName);
                var channel = new NotificationChannel(channelId, channelNameJava,
NotificationImportance.Default)
                {
                    Description = channelDescription
                };
                manager.CreateNotificationChannel(channel);
            }
            channelInitialized = true;
        }
        /// <summary>
        /// Десериализация оповещения.
        /// </summary>
        /// <param name="notificationString"></param>
        /// <returns></returns>
        private LocalNotification DeserializeNotification(string notificationString)
        {
            var xmlSerializer = new XmlSerializer(typeof(LocalNotification));
            using (var stringReader = new StringReader(notificationString))
            {
                var notification =
(LocalNotification)xmlSerializer.Deserialize(stringReader);
                return notification;
            }
        }
```

```
    }
}
```

2.2.2 MyFirebaseAuth.cs

```
using System.Threading.Tasks;
using appUI.Droid.Persistents;
using appUI.Persistents;
using Firebase.Auth;
using Xamarin.Forms;

[assembly: Dependency(typeof(MyFirebaseAuth))]

namespace appUI.Droid.Persistents
{
    /// <summary>
    /// Класс для регистрации\авторзации пользователей.
    /// </summary>
    public  class MyFirebaseAuth : IFirebaseAuth
    {
        /// <summary>
        /// Авторизация пользвателя.
        /// </summary>
        /// <param name="email">Электронная почта.</param>
        /// <param name="password">Пароль.</param>
        /// <returns>Токен</returns>
        public async Task<string> LoginWithEmailPassword(string email, string password)
        {
            try
            {
                var user = await
FirebaseAuth.Instance.SignInWithEmailAndPasswordAsync(email, password);
                var token = await user.User.GetIdTokenAsync(false);
                return token.Token;
            }
            catch
            {
                return "";
            }
        }
        /// <summary>
        /// Регистрация нового пользователя.
        /// </summary>
        /// <param name="email">Электронная почта.</param>
        /// <param name="password">Пароль.</param>
        /// <returns>Токен</returns>
        public async Task<string> CreateWithEmailPassword(string email, string password)
        {
            try
            {
                var user = await
FirebaseAuth.Instance.CreateUserWithEmailAndPasswordAsync(email, password);
```

```
                var token = await user.User.GetIdTokenAsync(false);
                return token.Token;
            }
            catch
            {
                return "";
            }
        }
        /// <summary>
        /// Определяет выполнил ли пользователь вход.
        /// </summary>
        /// <returns>Результат.</returns>
        public bool IsCurrentUser() => FirebaseAuth.Instance.CurrentUser != null;
        /// <summary>
        /// Возвращает уникальный идентификатор пользователя.
        /// </summary>
        public string GetUserID() => FirebaseAuth.Instance.CurrentUser.Uid;
        /// <summary>
        /// Выполняет выход учетной записи из системы.
        /// </summary>
        public void SignOut()
        {
            try
            {
                FirebaseAuth.Instance.SignOut();
            }
            catch {
                var notification = new LocalNotificationService();
                notification.LongAlert("Please check your internet connection.");
            }
        }
    }
}
```

2.2.3 SQLiteDb.cs

```
using System;
using System.IO;
using appUI.Droid.Persistents;
using appUI.Persistents;
using SQLite;
using Xamarin.Forms;

[assembly: Dependency(typeof(SQLiteDb))]
namespace appUI.Droid.Persistents
{
    /// <summary>
    /// Класс для получения доступа к локальной базе данных.
    /// </summary>
    public class SQLiteDb : ISQLite
    {
        /// <summary>
        /// Путь для сохранения файла базы данных.
        /// </summary>
        readonly string path =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        /// <summary>
        /// Возвращает соединение для работы с базой данных.
        /// </summary>
        /// <param name="id">Имя файла базы данныхю</param>
        /// <returns></returns>
        public SQLiteAsyncConnection GetConnectionPersonal(string id) =>
            new SQLiteAsyncConnection(Path.Combine(path, id +".db3"));
    }
}
```

Список используемой литературы

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

Лист регистрации изменений.

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	Новых	аннулированных					