

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра программной инженерии
Специальность 1-40 01 01 Программное обеспечение информационных технологий
Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет приложений)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА:

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема «Приложение для турнирной таблицы Enjoy Hockey League»

Исполнитель
студент (ка) 2 курса группы 4 _____ Горощеня Владислав Сергеевич
(Ф.И.О.)

Руководитель работы _____ ассистент Чистякова Ю.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____
 Председатель _____ Пацей Н.В.
 (подпись)

Минск 2023

Содержание

Введение	4
1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному продукту	5
1.1 Анализ прототипов	5
1.1.1 Kontinental Hockey League	5
1.1.2 NHL	7
1.1.3 Liiga	11
1.2 Формирование требований к программному средству	13
2 Моделирование предметной области и разработка функциональных требований	14
2.1 Описание средств разработки	14
2.2 Определение функциональных требований к программному средству	15
3 Проектирование программного средства	16
3.1 Общая структура проекта	16
3.2 Проектирование базы данных	19
3.3 Взаимоотношения между классами	22
3.4 Проектирование архитектуры	22
3.5 Проектирование последовательностей	23
4 Реализация программного средства	24
4.1 Реализация паттернов	24
4.2 Command	24
4.3 Реализация регистрации и авторизации пользователей	25
4.4 Реализация главного окна	26
4.4.1 Клиентская часть	26
4.4.1.1 Страница с матчами	26
4.4.1.2 Страница с составами	26
4.4.1.3 Страница со статьями	26
4.4.1.4 Страница с бонусами	26
4.4.2 Администраторская часть	27
4.4.2.1 Страница с матчами	27
4.4.2.2 Страница с составами	27
4.4.2.3 Страница со статьями	27
4.4.2.4 Страница с бонусами	28
5 Тестирование, проверка работоспособности и анализ полученных результатов	29
5.1 Тестирование авторизации и регистрации	29
5.2 Тестирование изменения результата матчей	31
5.3 Тестирование изменений в составе команды	34
5.4 Тестирование добавления статьи и бонуса	38
6 Руководство по установке и использованию	39
6.1 Руководство по установке	39

6.2 Руководство по пользованию	39
Заключение	43
Список использованных источников.....	44
ПРИЛОЖЕНИЕ А.....	45
ПРИЛОЖЕНИЕ Б	47
ПРИЛОЖЕНИЕ В.....	48
ПРИЛОЖЕНИЕ Г	50
ПРИЛОЖЕНИЕ Д.....	61
ПРИЛОЖЕНИЕ Е	63
ПРИЛОЖЕНИЕ Ж.....	65
ПРИЛОЖЕНИЕ З.....	66
ПРИЛОЖЕНИЕ И.....	67
ПРИЛОЖЕНИЕ К.....	68
ПРИЛОЖЕНИЕ Л.....	71
ПРИЛОЖЕНИЕ М.....	74

Введение

Хоккей – один из самых захватывающих видов спорта, имеющий довольно широкую аудиторию по всему миру. Данный вид спорта появился в Канаде, а первый хоккейный матч был сыгран в 1875 в Монреале на озере Виктория. Он дико популярен не только в Америке, но и Евразии тоже. Следует выделить такие страны как Финляндия, Швеция, Чехия и Словакия, которые неоднократно становились победителями различных мировых первенств. Для поддержания интереса к хоккею не только внутри страны, но и в мире, создаются хоккейные лиги – турниры среди местных или же приглашённых команд, что позволяет фанатам активно следить за успехами и развитием своей любимой команды.

Одной из важнейших отображений данных критериев является турнирная таблица, где представляется количество сыгранных матчей, количество забитых и пропущенных шайб, выигранных и проигранных матчей, а также очков. Данные пункты играют очень важную роль в развитии событий в лиге и позволяют держать интригу среди болельщиков различных команд.

Данное программное устройство позволяет наглядно отобразить турнирную таблицу вымышленной лиги Enjoy Hockey League с возможностью сортировки по различным статистическим показателям. Авторизованный пользователь также сможет просмотреть составы команд и статистику её игроков. Для получения более подробной информации о происходящем, он сможет почитать интересующие его статьи. Также в качестве бонуса, он сможет получить скидку на атрибутику или же билет на матч. Помимо этого, для прогнозирования изменений в турнирной таблице, он может посмотреть матчи команды, а также просмотреть протокол предыдущих матчей, например, если пользователь не сумел его посмотреть.

Также оно незаменимо с точки зрения административной части. Администратор может вносить изменения в результаты игр, менять составы, добавлять, удалять и править статьи, а также добавлять новые бонусы для пользователя и иметь полный доступ ко всем заказам.

Главная задача данного курсового проекта – разработка приложения, реализующего выше перечисленные функции и решающее поставленные задачи, с помощью языка программирования C#, основных приёмов и принципов объектно – ориентированного программирования, использования SQL Server Management Studio для создания баз данных и технологии Windows Presentation Forms, позволяющей создавать графические интерфейсы, основываясь на векторной графике, ориентированные на разработку клиентских Windows – приложений.

1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному продукту

В качестве источников литературы используются статьи с таких электронных ресурсов как <https://metanit.com/>, <https://proffesorweb.ru/>, официальной документации <https://docs.microsoft.com/en-us/dotnet/csharp/>, статьи с <https://habr.com/ru/>, а также курсы лекций по предмету «Объектно-ориентированные технологии программирования и стандарты проектирования» и «Базы данных».

1.1 Анализ прототипов

1.1.1 Kontinental Hockey League

Самым популярным приложением на территории СНГ данного типа является приложение Континентальной Хоккейной Лиги. Пользователь имеет право им пользоваться по исполнению 13 лет. У приложения довольно неплохой интерфейс – приятная цветовая гамма и шрифт. Навигация находится в бургер меню, что позволяет удобно ориентироваться по страницам. Помимо этого, в приложении имеется собственный магазин, где пользователь может купить атрибутику. Данный критерий приведён на рисунке 1.1.

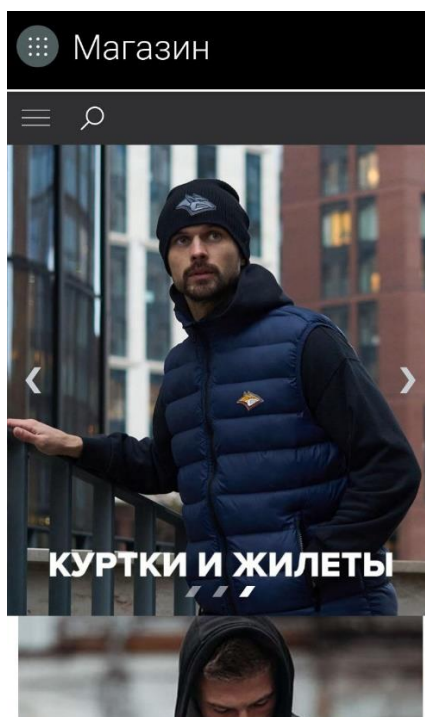


Рисунок 1.1 – магазин приложения КХЛ

Турнирная таблица содержит в себе общее число матчей, количество побед в основное время, в овертайме, по буллитам и поражений по тому же принципу, а также соотношение забитых и пропущенных и, последним, общее число очков.


Помимо этого, команда, вышедшая в плей-офф, выделяется жирным шрифтом. В мобильной версии ввиду очень широкой колонки с названием команды, данные приходится пролистывать, что является не совсем удобным. Отображение таблицы приведено на рисунке 1.2.



РЕГУЛЯРНЫЙ ЧЕМПИОНАТ 2022/2023				
КОНФЕРЕНЦИИ		ДИВИЗИОНЫ	ЧЕМПИОНАТ	
Место		Игры	В	Очки
1	СКА	68	40	105
2	ЦСКА	68	33	94
3	Локомотив	68	35	92
4	Ак Барс	68	27	91
5	Торпедо	68	31	90
6	Динамо М	68	29	87
7	Салават Юлаев	68	28	86
8	Авангард	68	27	86
9	Автомобилист	68	30	83
10	Металлург Мг	68	30	83
11	Сибирь	68	21	83
12	Витязь	68	24	76

Рисунок 1.2 – турнирная таблица Kontinental Hockey League

Календарь матчей можно разделить на 2 составляющие: календарь команды, за которой следит пользователь, и общий. По нажатию кнопки с надписью мой любимый клуб, можно менять его представление. Матчи отсортированы по времени: ближайшие вверху, прошедшие внизу. Причём в прошедших матчах идёт разделение по голам за период. Данный критерий приведён на рисунке 1.3.



СЕГОДНЯ: 31.03.23	
Все матчи: Динамо Мн Сбросить фильтр	
11.03.23	
Динамо Мн	СКА
2:4	
1:2 1:1 0:1	
Счет в серии: 2:4	
09.03.23	
СКА	Динамо Мн
6:3	
2:1 1:0 3:2	
Счет в серии: 3:2	
07.03.23	

Рисунок 1.3 – календарь матчей

Составы команд также представлены в виде таблицы, причём перед этим идёт общая фотография состава команды. В строках таблицы приводятся номер

игрока, его фотография, имя фамилия и страна. Для подробного изучения, можно перейти на соответствующую строку. Данные приведены на рисунках 1.4.

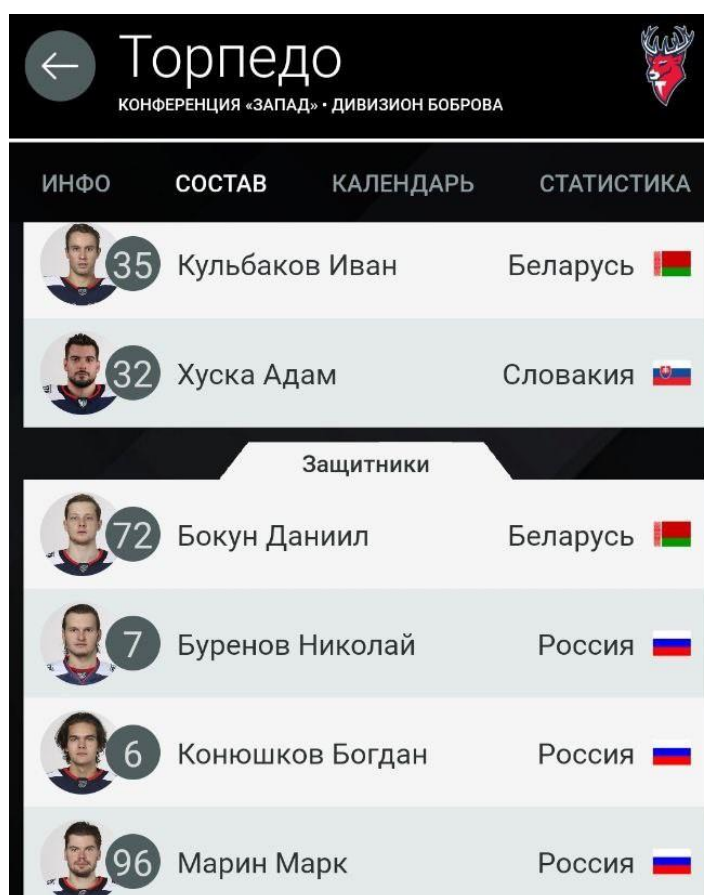


Рисунок 1.4 – представление состава команды

Важно отметить, что иногда в данном приложении при просмотре состава происходит ситуация, когда имя и фамилия очень длинные, и флаг страны не виден.

1.1.2 NHL

NHL – самая известная хоккейная лига в мире. Мечта каждого хоккеиста – попасть в данную лигу и прикоснуться к самому первому трофею в истории данного вида спорта – Кубку Стэнли. Она известна во всём мире. Однако в СНГ приложение данной лиги пользуется меньшей популярностью, чем предыдущий аналог. Данное приложение имеет довольно приятный и минималистичный дизайн в синих, чёрных и белых тонах. Можно выбрать тёмную и светлую темы. При начале пользования, пользователь сначала выбирает свою любимую команду, а затем он может выбрать другие команды, за которыми ему интересно следить. Их можно выбрать сколько угодно. Главная страница приложения включает в себя навигационную панель внизу, где есть секция с информацией о событиях в любимой команде, последние новости из лиги, результаты игр, турнирную таблицу, и прочее, где содержаться ссылки на официальные сайты магазина лиги, сайты лиги на разных языках и

настройки самого приложения. Секция с любимой командой включает в себя календарь её игр, обзор матчей и результаты встреч, новости, где она упоминается, а также сведения о лучших игроках по очкам, голам, передачам, общего времени на льду и показателе полезности. Помимо этого, в разделе More можно посмотреть состав команды и ссылки на соцсети.

Данная секция приведена на рисунке 1.6.

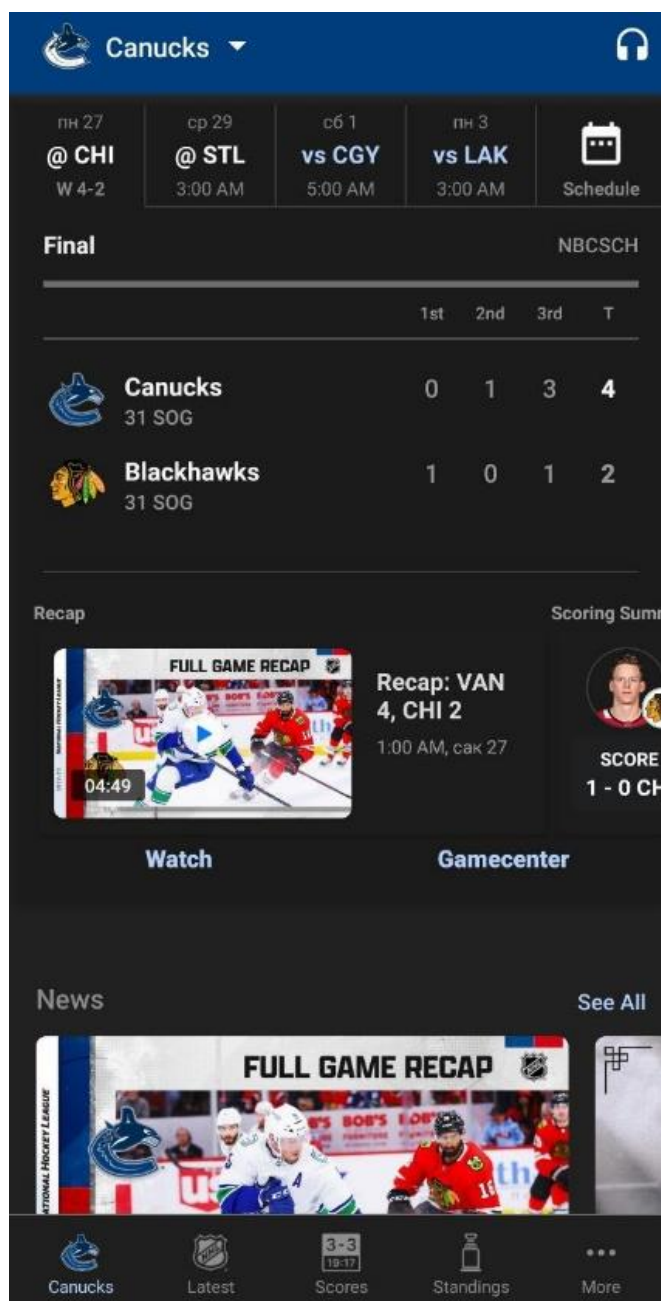


Рисунок 1.6 - Приложение NHL с секцией любимой команды

Календарь игр любимой команды представлен в виде классического календаря, причём домашние игры выделены синим, а гостевые – никак не выделяются. В ячейках с днями игр представлены аббревиатура названия команды соперника и

результат: выигрыш или проигрыш и счёт. Данный критерий представлен на рисунке 1.7.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2 MIN L 1-2	3	4 TOR W 4-1
5	6 NSH W 4-3	7	8 ANA W 3-2	9	10	11 OTT W 5-2
12	13	14 DAL W 5-2	15	16 ARI L 2-3	17	18 LAK W 3-2
19 ANA W 2-1	20	21 VGK L 3-4	22	23 SJS W 7-2	24	25 DAL W 3-1
26 CHI W 4-2	27	28 STL 3:00AM	29	30	31 CGY 5:00AM	

Home Away All times GMT+3:00 and subject to change

Рисунок 1.7 – Календарь игр любимой команды

Общий календарь имеет страничный вид. Можно выбрать определённый день и посмотреть результаты матчей за это время. Результаты матчей имеют вид блоков с логотипом и названием, причём под названием указано количество бросков, напротив каждой команды стоит число голов. Надпись с результатом, также может содержать подпись SO (дополнительное время). Общий календарь представлен на рисунке 1.8.

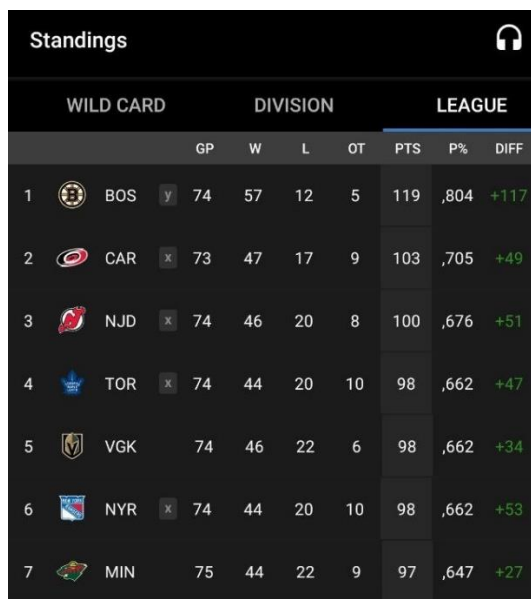
Final / SO		MSG-B
Canadiens 41 SOG	4	
Sabres 33 SOG	3	

Final		BSFL
Panthers 34 SOG	2	
Senators 27 SOG	5	

Рисунок 1.8 – календарь игр на определённый день

Турнирная таблица содержит в себе общее число игр, количество побед, поражений и поражений в дополнительное время, общее число очков, процент общего

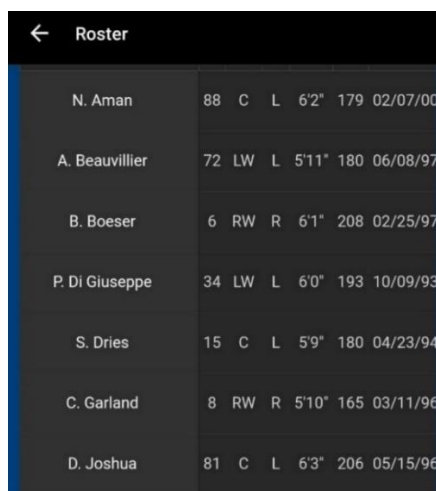
набранного количества от возможных, и разницу между забитыми и пропущенными. Команды, вышедшие в плей – офф помечаются крестиком напротив названия. Таблица представлена на рисунке 1.9.



Standings										
WILD CARD				DIVISION			LEAGUE			
				GP	W	L	OT	PTS	P%	DIFF
1		BOS	y	74	57	12	5	119	,804	+117
2		CAR	x	73	47	17	9	103	,705	+49
3		NJD	x	74	46	20	8	100	,676	+51
4		TOR	x	74	44	20	10	98	,662	+47
5		VGK		74	46	22	6	98	,662	+34
6		NYR	x	74	44	20	10	98	,662	+53
7		MIN		75	44	22	9	97	,647	+27

Рисунок 1.9 – турнирная таблица NHL

Составы различных команд посмотреть нельзя, только любимой команды. Как уже и упоминалось, это можно просмотреть в разделе «More». Он также представлен в виде таблицы, и здесь имеется разделение по позициям. В качестве колонок идут имя и фамилия игрока, его номер, рост и вес, дата рождения, место рождения. Представление состава показано на рисунке 1.10.



← Roster							
N. Aman	88	C	L	6'2"	179	02/07/00	
A. Beauvillier	72	LW	L	5'11"	180	06/08/97	
B. Boeser	6	RW	R	6'1"	208	02/25/97	
P. Di Giuseppe	34	LW	L	6'0"	193	10/09/93	
S. Dries	15	C	L	5'9"	180	04/23/94	
C. Garland	8	RW	R	5'10"	165	03/11/96	
D. Joshua	81	C	L	6'3"	206	05/15/96	

Рисунок 1.10 – представление состава

Для более подробного изучения данных игрока можно нажать на соответствующую строку.

1.1.3 Liiga

Liiga – главная хоккейная лига Финляндии, пусть и не самая известная на территории стран постсоветского пространства. Данная лига также имеет собственное приложение, и во многом оно схоже с приложением NHL. Например, меню представляет собой панель внизу, но прочие элементы для навигации открываются через бургер – меню. Здесь имеются разделы о последних новостях лиги, турнирную таблицу, расписание игр, а также составы команд.

Турнирная таблица включает в себя общее число игр, число побед, ничей, поражений, побед в дополнительное время и очков. Таблица представлена на рисунке 1.11.

22:13 | 0,0KB/c

48

Sarjataulukko

Tilastot









1	 Lukko	54	27	12	15	7	100
2	 Ilves	52	29	8	15	4	99
3	 Tappara	52	27	11	14	6	98
4	 Pelicans	53	27	7	19	3	91
5	 Kärpät	53	23	15	15	6	90
6	 KalPa	52	24	11	17	5	88
7	 HIFK	53	21	14	18	7	84
8	 TPS	52	19	16	17	9	82

Рисунок 1.11 – Таблица Liiga

Можно просмотреть составы любых команд. Здесь также имеется разделение по позициям, однако составы представлены уже не в виде таблиц, а в виде блоков с фотографией, номером и именем игрока. Для более подробной информации можно также перейти в блок определённого игрока.

Представление состава показано на рисунке 1.12.

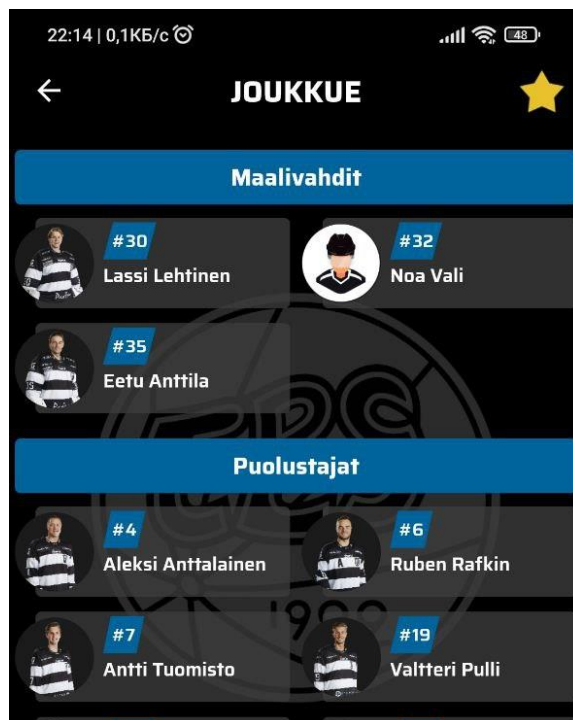


Рисунок 1.12 – представление состава команды

Календарь также представлен в виде списка, отсортированного по времени. Здесь есть разделение по месяцам, а с помощью дополнительного меню, можно выбрать матчи определённой команды. Календарь представлен на рисунке 1.13.

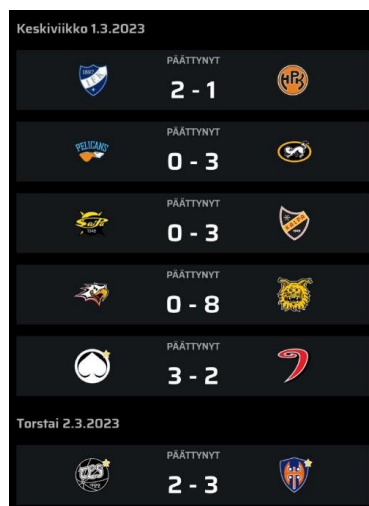


Рисунок 1.13 – представление календаря

Важно отметить, что несмотря на наличие на официальном сайте двух языков, в мобильной и десктопной версиях данный механизм не поддерживается, из-за чего проблематично ориентироваться в нём пользователям, не знающим финского.

1.2 Формирование требований к программному средству

Обзор вышеперечисленных аналогов позволяет сформировать на основе собственных преимуществ и недостатков список требований к разрабатываемому программному продукту. Программное средство необходимо выполнять следующие функции:

- управление базой данных администратором;
- возможность пользователем регистрироваться в данном приложении и войти в существующую запись;
- возможность пользователем просматривать турнирную таблицу, составы команд и результаты матчей;
- предоставление администратором бонусов для пользователя и возможность пользователем пользоваться им;
- поддержка обратной связи.

Важно отметить, что проект необходимо скомпилировать и протестировать, проанализировать его работу и возможность, внести определённые изменения в код и требования с целью возможности его дальнейшего развития.

2 Моделирование предметной области и разработка функциональных требований

2.1 Описание средств разработки

Microsoft Visual Studio 2019 – интегрированная среда разработки, предназначенная для написания, отладки и сборки кода, позволяющая создавать не только консольные, но и десктопные приложения с помощью таких технологий как Window Presentation Forms или WinForms.

Язык программирования С# – объектно - ориентированный язык программирования с Си – подобным синтаксисом и поддержкой строгой типизации, перегрузки операторов, а также атрибутов, событий, свойств и исключений. Является основным языком .NET Framework и технологии WPF.

Технология Windows Presentation Forms – платформа уровня представления для построения графических интерфейсов, основывающаяся на векторной системе визуализации и ориентированная на разработку клиентских Windows – приложений. Данная технология позволяет взаимодействовать с пользователем, благодаря средствам языка XAML. Её графической составляющей является технология DirectX, выигрывающая по производительности у GDI/GDI+, используемой WinForms, за счёт использования аппаратного ускорения графики.

Расширяемый язык разметки XAML (eXtensible Application Markup Language) – язык декларативного описания интерфейса, основанный на XML. Он позволяет совмещать работу дизайнера и разработчика с помощью таких инструментов как элементы управления, привязки данных, стилей, шаблонов, макетов, анимаций, а также двумерной и трёхмерной график. Важно отметить, что XAML компилируется в BAML – своё бинарное представление, оптимизированное для времени выполнения и внедряющееся в ресурсы сборки.

Entity Framework – объектно - ориентированная технология доступа к базам данных от Microsoft, позволяющая взаимодействовать с СУБД с помощью сущностей. Сущность представляет собой набор данных, ассоциированный с определённым объектом. EF позволяет работать с данными независимо от типа хранилища, тем самым абстрагировавшись от базы данных. Она предоставляет набор классов, через которые можно устанавливать подключения, отправлять запросы к базам данных и получать результат, а также совершать ряд иных операций. Существует три способа взаимодействия с БД для Entity Framework:

- Code first: сначала пишется код на языке программирования С#, и исходя из него формируется база данных.

- Database first: сначала создаётся база данных, а затем генерируется модель базы данных.

- Model first: сначала создаётся графическая модель базы данных, и в это же время создаются классы С# в фоновом режиме. После чего, генерируется база данных на основе данной диаграммы.

Microsoft SQL Server – наиболее известная система управления базами данных, обладающая такими характеристиками как высокая производительность, надёжность и безопасность с предоставлением шифрования данных, а также простотой администрирования. Она представляет собой реляционную модель, где взаимодействие с таблицами происходит исходя из теории множеств, а таблицы представляют совокупность строк и столбцов, где каждая строка хранит в себе объект, а столбцы – его атрибуты. Для взаимодействия с базой данных используется язык структурированных запросов SQL. Принцип его работы выглядит следующим образом: пользователь (или же клиент) посылает запрос, тот определённым образом интерпретируется и результат посылается пользователю. Transact-SQL – наиболее распространённый язык запросов, включающий в себя управляющие операторы, локальные и глобальные переменные, а также различные дополнительные функции для дат, математических вычислений и обработки строк.

2.2 Определение функциональных требований к программному средству

Разрабатываемое приложение необходимо предоставлять следующий функционал для пользователя:

- возможность регистрации;
- возможность авторизации;
- возможность выбора любимой команды;
- просмотр турнирной таблицы;
- просмотр результатов матчей;
- просмотр составов команд;
- пользование бонусом;
- возможность смены языка и темы;
- просмотр статей;
- сортировка данных в таблице.

Разрабатываемое приложение необходимо предоставлять следующий функционал для администратора:

- возможность авторизации;
- просмотр и обновление турнирной таблицы;
- просмотр и обновление результатов матчей;
- просмотр составов команд;
- предоставление бонуса клиенту;
- просмотр и добавление статей;
- возможность поддержки связи с пользователем.

3 Проектирование программного средства

3.1 Общая структура проекта

Описание общей структуры основных файлов проекта и папок приведено в таблице 3.1.

Таблица 3.1 – описание структуры папок и файлов проекта

Имя файла	Содержание
App.config	Файл с параметрами проекта.
Папка Models	Папка, содержащая описание используемых в приложении данных
Папка Icons	Папка, содержащая все необходимые изображения.
Папка ViewModels	Папка, содержащая описание механизмов связывания модели данных с представлением.
Папка Styles	Папка, содержащая словари ресурсов для стилей.
AddingPlayer.xaml	Окно для добавления игрока администратором.
AdminAllOrders.xaml	Окно для просмотра администратором всех используемых пользователем заказов.
AdminArticles.xaml	Страница администратора для работы со статьями.
AdminBonus.xaml	Страница администратора для работы с бонусами.
AdminLookRosters.xaml	Окно администратора для работы с составами.
AdminMatches.xaml	Страница администратора для работы с результатами матчей
AdminRosters.xaml	Страница администратора для просмотра команд
Article.xaml	Страница пользователя для просмотра статей
BelarussianDictionary.xaml	Словарь ресурсов, содержащий белорусский язык
Bonus.xaml	Страница пользователя для просмотра и выбора бонуса
EnglishDictionary.xaml	Словарь ресурсов, содержащий английский язык
Login.xaml	Страница для выполнения входа.

Продолжение таблицы 3.1

LookRosters.xaml	Страница пользователя для просмотра команд и их составов.
MainFrame.xaml	Основное окно с функционалом.
MainWindow.xaml	Основное окно приложения для логина и регистрации.
Matches.xaml	Страница пользователя с просмотром результатов матчей
Rosters.xaml	Страница пользователя для просмотра команд.
RussianDictionary.xaml	Словарь ресурсов, содержащий русский язык
Table.xaml	Страница для просмотра положения команд в турнирной таблице.
UpdatingMatch.xaml	Окно администратора для обновления результата матчей.
UpdatingPlayer.xaml	Окно администратора для изменения данных игрока

Более подробное описание содержимого папок представлено на рисунке 3.1.

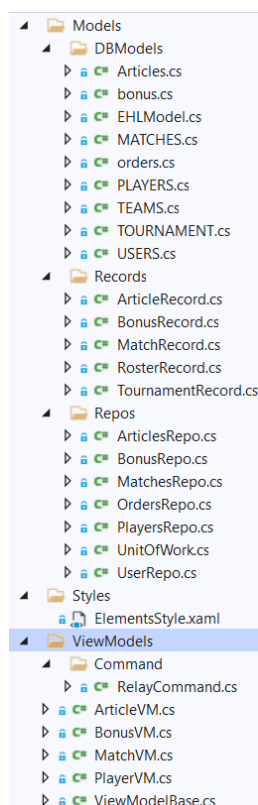


Рисунок 3.1 – подробное содержимое папок проекта

Описание структуры содержимого папок проекта приведено в таблице 3.2.

Таблица 3.2 – Описание содержимого папок проекта

Имя файла	Содержимое
Articles.xaml	Класс сущности Articles
bonus.cs	Класс сущности bonus
EHLModel.cs	Класс контекста подключения к базе данных
MATCHES.xaml	Класс сущности MATCHES
orders.xaml	Класс сущности orders
PLAYERS.xaml	Класс сущности PLAYERS
TEAMS.xaml	Класс сущности TEAMS
TOURNAMENT.cs	Класс сущности TOURNAMENT
USERS.cs	Класс сущности USERS
ArticleRecord.cs	Класс для отслеживания изменений сущности Articles
BonusRecord.cs	Класс для отслеживания изменений сущности Bonus
MatchRecord.cs	Класс для отслеживания изменений сущности MATCHES
RosterRecord.cs	Класс для отслеживания изменений сущности PLAYERS
TournamentRecord.cs	Класс для отслеживания изменений сущности TOURNAMENT
ArticleRepo.cs	Класс доступа к данным для сущности Articles
BonusRepo.cs	Класс доступа к данным для сущности bonus
MatchesRepo.cs	Класс доступа к данным для сущности MATCHES
OrdersRepo.cs	Класс доступа к данным для сущности orders
PlayersRepo.cs	Класс доступа к данным для сущности PLAYERS
UnitOfWork.cs	Класс, совмещающий контекст для остальных классов в соответствующей папке
UsersRepo.cs	Класс доступа к данным для сущности USERS
RelayCommand.cs	Класс для управления работой команд
ViewModelBase.cs	Класс содержащий переопределение метода OnPropertyChanged для возможности изменения полей сущности

Продолжение таблицы 3.2

ArticleVM.cs	Класс для получения из представлений для сущности Article
BonusVM.cs	Класс для получения из представлений для сущности bonus
MatchVM.cs	Класс для получения из представлений для сущности MATCHES
PlayerVM.cs	Класс для получения из представлений для сущности PLAYERS

3.2 Проектирование базы данных

Проектирование базы данных – процесс создания базы данных и определения ограничений целостности.

При проектировании БД использовалась объектно – ориентированная технология Entity Framework, а именно подход, объединяющий в себе подходы Database First и Code First. Обычно его называют Code Second. Для этого была создана база данных TEST2 с помощью системы управления реляционными базами данных Microsoft SQL Server. Она состоит из таблиц, приведённых на рисунке 3.2.



Рисунок 3.2 – схема базы данных TEST2

Структура таблицы USERS проиллюстрирована на рисунке 3.3.

Column Name	Data Type	Allow Nulls
id	uniqueidentifier	<input type="checkbox"/>
NAME	nvarchar(50)	<input checked="" type="checkbox"/>
SURNAME	nvarchar(50)	<input checked="" type="checkbox"/>
PASSWORD	nvarchar(50)	<input checked="" type="checkbox"/>
ROLE	int	<input checked="" type="checkbox"/>
TELEPHONE	nvarchar(13)	<input checked="" type="checkbox"/>
[Favourite team]	int	<input checked="" type="checkbox"/>

Рисунок 3.3 – структура таблицы USERS

В данной таблице поле ID, представленное 16-битным Guid-идентификатором, является первичным ключом. Поле NAME хранит имя

пользователя, SURNAME – фамилию, PASSWORD – пароль, TELEPHONE — номер телефона, NAME, ROLE – роль (1 для администратора, остальные для пользователя), FAVOURITE TEAM – идентификатор любимой команды.

В таблице TEAMS поле teamid – идентификатор команды, Logo – явный путь к логотипу команды, Team Name – название команды.

Структура таблицы TEAMS представлена на рисунке 3.4.

	Column Name	Data Type	Allow Nulls
🔑	teamid	int	<input type="checkbox"/>
	Logo	nvarchar(MAX)	<input checked="" type="checkbox"/>
	[Team Name]	nvarchar(50)	<input checked="" type="checkbox"/>

Рисунок 3.4 – структура таблицы TEAMS

Структура таблицы MATCHES представлена на рисунке 3.5. В ней поле matchid – идентификатор матча, GAME_DATE – дата игры, поля team1 и team2 – идентификатор команды хозяев и гостей соответственно, Logo1 и Logo2 – явный путь к их логотипам, а Goals1 и Goals2 – количество голов, которые та или иная команда забросила в данном матче.

	Column Name	Data Type	Allow Nulls
▶	Team1	int	<input checked="" type="checkbox"/>
	Logo1	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Goals1	int	<input checked="" type="checkbox"/>
	Goals2	int	<input checked="" type="checkbox"/>
	Logo2	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Team2	int	<input checked="" type="checkbox"/>
	GAME_DATE	datetime	<input checked="" type="checkbox"/>
🔑	matchid	int	<input type="checkbox"/>

Рисунок 3.5 – структура таблицы MATCHES

Структура таблицы TOURNAMENT проиллюстрирована на рисунке 3.6. В ней поле teamid – идентификатор команды, Logo – её логотип, Team Name – название, W – количество побед, T – количество ничей, L – количество поражений, P – общее количество очков.

	Column Name	Data Type	Allow Nulls
▶	teamid	int	<input type="checkbox"/>
	Logo	nvarchar(MAX)	<input checked="" type="checkbox"/>
	[Team Name]	nvarchar(50)	<input checked="" type="checkbox"/>
	W	int	<input type="checkbox"/>
	T	int	<input type="checkbox"/>
	L	int	<input type="checkbox"/>
	P	int	<input type="checkbox"/>

Рисунок 3.6 – структура таблицы TOURNAMENT

В таблице PLAYERS поле teamID – идентификатор команды, в которую игрок входит, Number – номер игрока в команде, Photo – путь к изображению с

игроком, Name – его имя, PositionID – позиция, Country – страна, PlayerId – его уникальный идентификатор, к которому мы можем обратиться при выполнении администратором определённых манипуляций, Height и Weight – рост и вес соответственно.

Структура таблицы PLAYERS приведена на рисунке 3.7.

	Column Name	Data Type	Allow Nulls
▶	teamID	int	<input type="checkbox"/>
	Number	int	<input type="checkbox"/>
	Photo	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Name	nvarchar(MAX)	<input checked="" type="checkbox"/>
	PositionID	nvarchar(50)	<input checked="" type="checkbox"/>
	Country	nvarchar(50)	<input checked="" type="checkbox"/>
🔑	PlayerId	uniqueidentifier	<input type="checkbox"/>
	Height	int	<input type="checkbox"/>
	Weight	int	<input type="checkbox"/>

Рисунок 3.7 – структура таблицы PLAYERS

Структура таблицы Articles приведена на рисунке 3.8. В ней поле id – идентификатор статьи, Header – заголовок, Image – путь к изображению, Creation Date – дата добавления статьи.

	Column Name	Data Type	Allow Nulls
▶ 🔑	id	uniqueidentifier	<input type="checkbox"/>
	Header	nvarchar(50)	<input checked="" type="checkbox"/>
	Image	nvarchar(50)	<input checked="" type="checkbox"/>
	[Creation Date]	date	<input checked="" type="checkbox"/>

Рисунок 3.8 – структура таблицы Articles

Структура таблицы bonus приведена на рисунке 3.9. В ней поле id – идентификатор бонуса, Header – название бонуса, Image – путь к изображению.

	Column Name	Data Type	Allow Nulls
▶ 🔑	id	uniqueidentifier	<input type="checkbox"/>
	header	nvarchar(MAX)	<input checked="" type="checkbox"/>
	image	nvarchar(MAX)	<input checked="" type="checkbox"/>

Рисунок 3.9 – структура таблицы bonus

Структура таблицы orders проиллюстрирована на рисунке 3.10.

	Column Name	Data Type	Allow Nulls
▶	id	uniqueidentifier	<input type="checkbox"/>
	userid	uniqueidentifier	<input type="checkbox"/>
	STATUS	nvarchar(32)	<input type="checkbox"/>

Рисунок 3.10 – структура таблицы orders

В ней поле id – идентификатор бонуса, userid – идентификатор пользователя, STATUS – текстовое поле, отражающее дату заказа.

3.3 Взаимоотношения между классами

Для представления взаимоотношений между классами используется диаграмма UML – графический набор элементов, представляющийся чаще всего связанным графом с вершинами, которые называют сущностями, и рёбрами, называемых отношениями.

Для представления внутренней структуры программы в виде классов и связей используется диаграмма классов. Она представлена в приложениях А и Б.

3.4 Проектирование архитектуры

Для общего представления функционального назначения системы используется диаграмма использования, описывающая доступный функционал каждой группе пользователей.

Диаграмма использования приведена на рисунке 3.11.

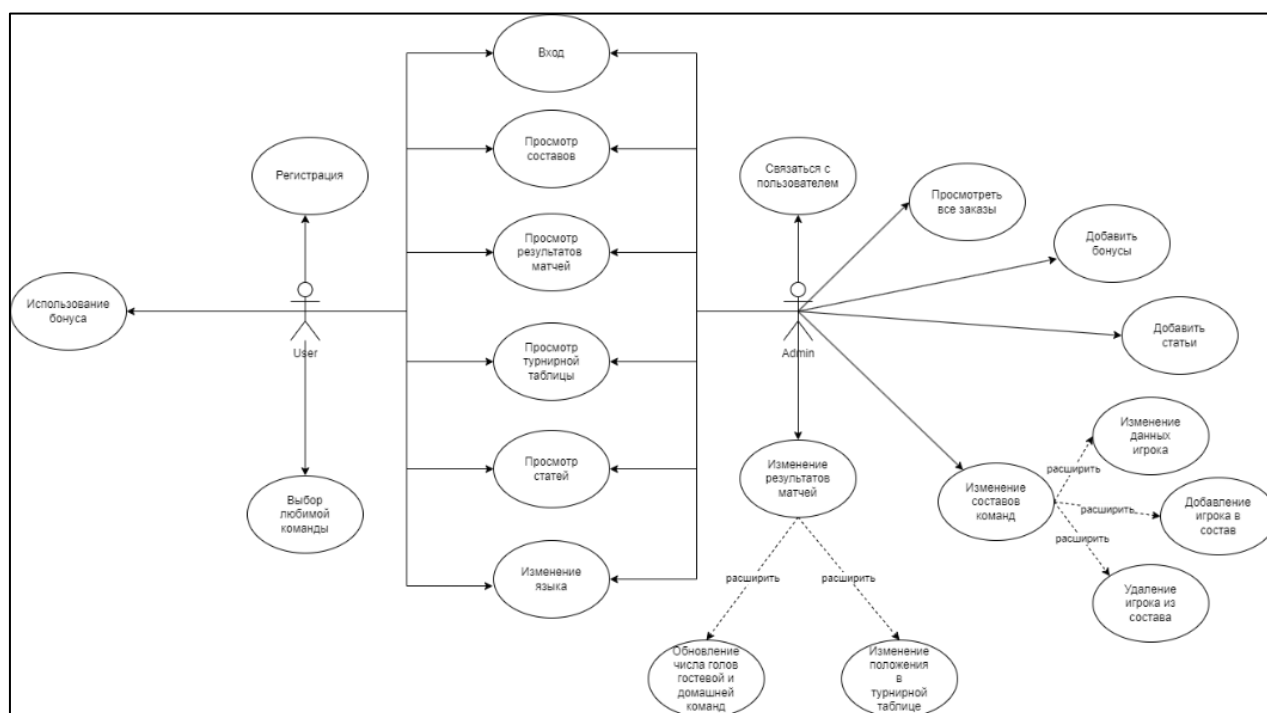


Рисунок 3.11 – диаграмма использования

В ней используются два типа сущностей: варианты использования, являющиеся любой функцией системы, и группы пользователей, обычно называемых актёрами. Каждый вариант использования обозначает набор действий, которые актёр может использовать для взаимодействия с системой, при этом определяя выполняемый системой набор действий.

3.5 Проектирование последовательностей

Визуализация взаимодействия объектов системы между собой во времени осуществляется с помощью диаграммы последовательностей, позволяющей представить её в едином сценарии. Она иллюстрирует взаимодействие друг с другом различных частей системы для выполнения определённых функций и порядок их выполнения. Время сценария отображается с помощью линии жизни объекта, показываемой штриховой линией, направленной вертикально вниз. Она показывает время существования объекта в системе, причём сами объекты показаны в виде прямоугольников, а сообщения, которыми они обмениваются, представляют собой линии со стрелкой. Диаграмма последовательностей представлена в приложении В.

4 Реализация программного средства

4.1 Реализация паттернов

Поскольку способ реализации программного проекта во многом схож с паттерном MVVM, включающий в себя три компонента, позволяющих упростить разработку и тестированию и дальнейшую модификацию, то в первую очередь необходимо определить базовые классы – модели, выражающие сущности, с которыми происходит взаимодействие.

Для реализации классов модели был создан класс `ViewModelBase`, реализующий интерфейс `INotifyPropertyChanged`. Для его реализации необходимо объявить свойство `PropertyChanged` и метод `OnPropertyChanged()`. Для отслеживания свойства, вызывается метод `OnPropertyChanged(«Имя свойства»)`. Реализация класса `ViewModelBase` представлено на рисунке 4.1.

```
Ссылка: 7
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    Ссылка: 35
    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Рисунок 4.1 – реализация класса `ViewModelBase`

Для удобства работы с данными, используется паттерн `Repository`, позволяющий абстрагироваться от конкретных подключений к источникам данных, являясь своего рода звеном между классами, взаимодействующих с данными, и остальной программой. Во избежание объявления отдельного класса репозитория для каждой модели базы данных, можно его обобщить, с помощью паттерна `UnitOfWork`, где указываются все обобщённые репозитории. Реализация паттерна `UnitOfWork` приведена в приложении Г.

4.2 Command

Команды предоставляют взаимодействие между пользователем и приложением, то есть это механизм выполнения какой – либо задачи. Они позволяют нам сократить объём кода и использовать их многократно для нескольких элементов управления в различных местах программы. Для их использования в проекте было необходимо добавить новый класс под названием `RelayCommand`, содержащий в себе методы `CanExecute` и `Execute`. `CanExecute` определяет, может ли выполняться команда, а `Execute` – собственно выполняет логику команды.

Реализация класса RelayCommand приведено на рисунке 4.2.

```
public class RelayCommand : ICommand
{
    private readonly Action<object> _execute;
    private readonly Predicate<object> _canExecute;

    Ссылка: 3
    public RelayCommand(Action<object> execute)
    {
        : this(execute, null)
    }

    Ссылка: 11
    public RelayCommand(Action<object> execute, Predicate<object> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }

    Ссылка: 0
    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }

    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }

    Ссылка: 0
    public void Execute(object parameter)
    {
        _execute(parameter);
    }
}
```

Рисунок 4.2 – реализация класса RelayCommand

Реализация команд необходимых для работы с данными приведены в приложении Д.

4.3 Реализация регистрации и авторизации пользователей

Для использования приложения каждый пользователь должен войти в систему, предварительно зарегистрировавшись.

При регистрации необходимо заполнить поля Имя, Фамилия, Телефон, Пароль и выбрать через Combobox любимую команду. По нажатию кнопки MoveToMainFrame срабатывает метод MoveToMainFrame_Click, при котором проверяется заполненность всех полей, а также корректность введённого номера телефона. Также важно отметить, что в данном классе содержится метод SecureStringToString(), преобразующий введённый пользователем пароль в строковое значение для последующей проверки на соответствие с сохранённым в базе данных паролем. Если всё заполнено корректно, то пользователь добавляется в базу данных, и он переходит на страницу с авторизацией. Реализация класса Registration приведена в приложении Е.

В ней необходимо ввести имя, фамилию и пароль. По нажатию кнопки MoveToMainFrame срабатывает метод MoveToMainFrame_Click, проверяющий наличие в базе данных введённых имени, фамилии и пароля. В успешном случае, окно с авторизацией закрывается, и открывается окно с контентом. Реализация класса Login приведена в приложении Ж.

4.4 Реализация главного окна

При переходе в главное окно открывается страница с турнирной таблицей, где команды отсортированы по количеству очков. Вверху каждой страницы имеется шапка с кнопкой для возвращения в окно с авторизацией. Внизу окна располагается панель, позволяющей переходить на различные страницы. Важно учитывать, что содержимое страницы зависит от роли пользователя.

4.4.1 Клиентская часть

4.4.1.1 Страница с матчами

Страница с результатами матчей содержит в себе список, состоящий из блоков, внутри которых располагается логотипы команды хозяев и гостей, причём хозяева находятся в левой части, а гостей – в правой соответственно. Напротив них располагается число голов, забитых данными командами в данном матче. Вверху располагается дата и время матча.

4.4.1.2 Страница с составами

Страница с составами содержит в себе список, состоящий из блоков с логотипом команды и её названием. При нажатии на блок срабатывает метод `ListViewItem_PreviewMouseLeftButtonDown`, который позволяет нам открыть отдельное окошко с составом команды, где находится таблица с номерами игроков, их фотографии, позиции, роста, веса и страны. Листинг реализации приведён в приложении 3.

4.4.1.3 Страница со статьями

Страница со статьями включает в себя список, состоящий из блоков с фотографией в левой их стороне, заголовка статьи и времени добавления статьи в базу данных.

4.4.1.4 Страница с бонусами

Страница с бонусами представляет собой список, состоящий из блоков с фотографией в левой их части, заголовка. Также под заголовком находится кнопка `useBonus` по нажатию на которую срабатывает команда `GetID`, в качестве параметра которой содержится идентификатор бонуса. Для работы команды используется специальный метод `ReturnID`, в котором происходит добавления заказа бонуса в базу данных, а администратору приходит уведомление о том, что пользователь воспользовался бонусом. Реализация приведена в приложении И.

4.4.2 Администраторская часть

4.4.2.1 Страница с матчами

Страница с результатами матчей содержит в себе список, состоящий из блоков, внутри которых располагается логотипы команды хозяев и гостей, причём хозяева находятся в левой части, а гостей – в правой соответственно. Напротив них располагается число голов, забитых данными командами в данном матче. Вверху располагается дата и время матча. Под счётом игры располагается кнопка для изменения результата матча. При её нажатии проверяется, был ли обновлён до этого матч. Если он был обновлён, то результаты текущего матча вычитаются из таблицы. Помимо этого, по нажатию открывается и окошко, содержащее в себе блок с матчем, где на месте числа голов располагаются 2 текстовых поля для их заполнения. При принятии изменений проверяется соотношение голов и результаты матча обновляются, причём изменения происходят как в списке с результатами матча, так и в турнирной таблице. Реализация приведена в приложении К.

4.4.2.2 Страница с составами

Страница с составами содержит в себе список, состоящий из блоков с логотипом команды и её названием. При нажатии на блок срабатывает метод `ListViewItem_PreviewMouseDown`, который позволяет нам открыть отдельное окошко с составом команды, где находится таблица с номерами игроков, их фотографии, позиции, роста, веса и страны, а также кнопки для обновления и удаления. Над таблицей также находится кнопка для добавления игрока в состав выбранной команды. При нажатии на эту кнопку открывается окно с формой, где заполняются: Имя, Номер, Позиция, Страна, Рост и Вес, а также при нажатии на кнопку с рамкой выбирается фотография. При нажатии кнопки `Добавить`, срабатывает метод `AddData` через команду `AddCommand`, имеющей в качестве параметра идентификатор команды, в которую мы добавляем игрока. После чего игрок добавляется в базу и отображается в окне с составом. При обновлении открывается такое же окно, и после обновления данных удаляется старая запись, которую заменяет новая имеющая такой же идентификатор. Листинг реализации приведён в приложении Л.

4.4.2.3 Страница со статьями

Страница со статьями имеет в наличии `GroupBox`, состоящий из кнопки, которая по нажатию позволяет открыть `OpenFileDialog` и выбрать изображение, и текстового поля, куда вводится заголовок статьи. По нажатию кнопки `Добавить` над `GroupBox` статья добавляется в базу данных, а добавление отображается в списке, аналогичном списку статей в клиентской части. Над заголовком отображается дата добавления статьи в базе данных. Листинг реализации приведён в приложении М.

4.4.2.4 Страница с бонусами

Страница со статьями имеет в наличии Groupbox, состоящий из кнопки, которая по нажатию позволяет открыть OpenFileDialog и выбрать изображение, и текстового поля, куда вводится заголовок бонуса. По нажатию кнопки Добавить над Groupbox бонус добавляется в базу данных, а добавление отображается в списке, аналогичном списку статей в клиентской части, но без кнопки использования бонуса. В шапке страницы помимо кнопки закрытия окна, имеется кнопка, позволяющая посмотреть все заказы пользователей. Листинг реализации приведён в приложении Н.

5 Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование является одной из важнейших частей разработки программного продукта, позволяющее убедиться в корректности работы приложения во время его эксплуатации. Очень важно его чётко организовать для обеспечения полного тестирования каждого взаимодействия пользователя с приложением. Тестирование должно включать в себя следующие пункты: подготовка тестовых данных, анализ результатов и регулярное тестирование.

Подготовка тестовых данных подразумевает под собой использование определённых данных для проверки каждого взаимодействия. Они должны воспроизводить реальные сценарии использования и включать в себя данные, позволяющие обнаружить недочёты в разработке, что позволяет перейти к анализу результатов. При выявлении проблем и недочётов, необходимо провести отладку приложения для поиска таких участков программы, что нарушают корректность её работы. Регулярное тестирование представляет собой постоянную проверку работоспособности приложения при изменении в коде или же изменении его окружения.

5.1 Тестирование авторизации и регистрации

При авторизации или регистрации возможна ситуация, когда пользователь вводит некорректные данные, такие как неверный пароль, незарегистрированные имя и фамилию, неверный формат номера телефона. Для уведомления его об этом появляются специальные окна с сообщениями об ошибке и её сути. Результаты представлены на рисунках 5.1 – 5.8.

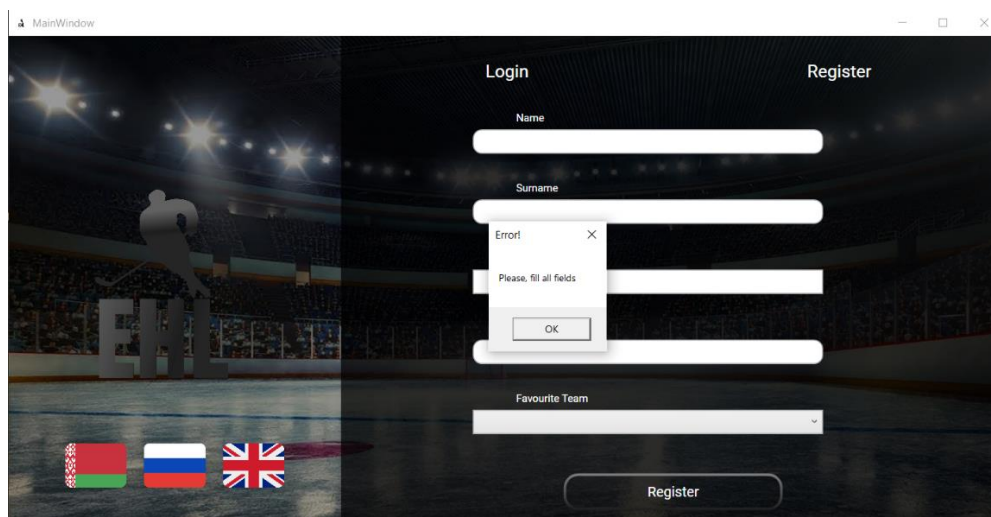


Рисунок 5.1 – обработка пустых полей

На рисунках 5.2 и 5.3 приведён пример тестирования корректности номера телефона. Регулярное выражение для номера телефона начинается с +375, после

идут числа, характеризующие номер оператора: 25, 29, 33, 44. И затем следует комбинация из 7 чисел.

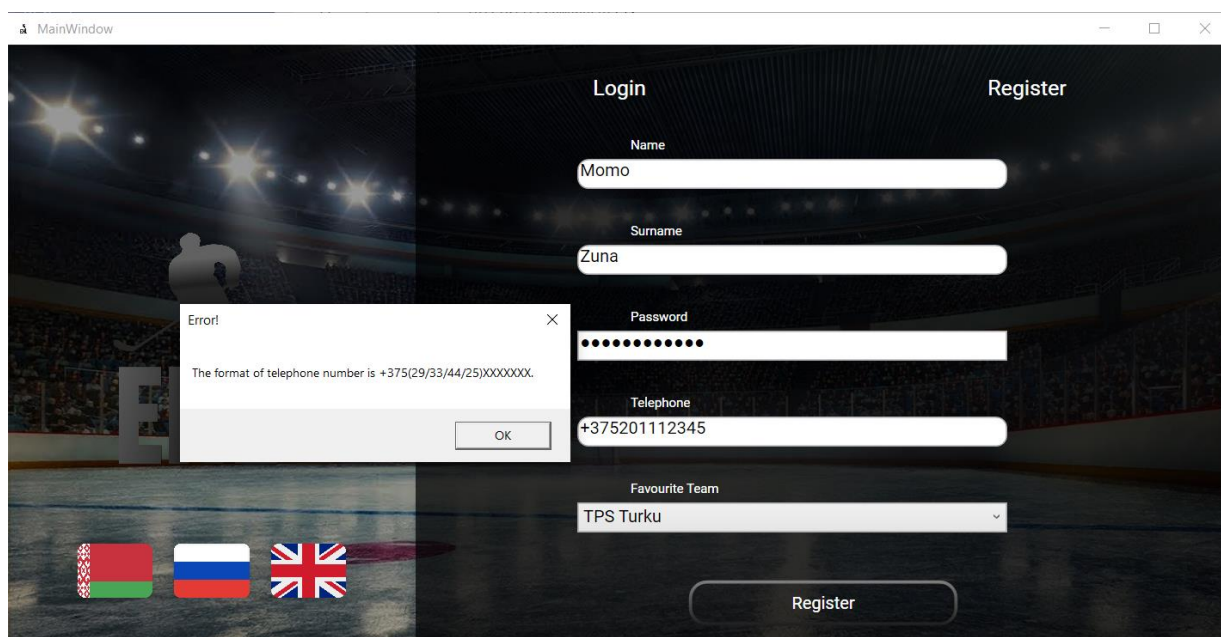


Рисунок 5.2 – обработка номера телефона с неверным оператором

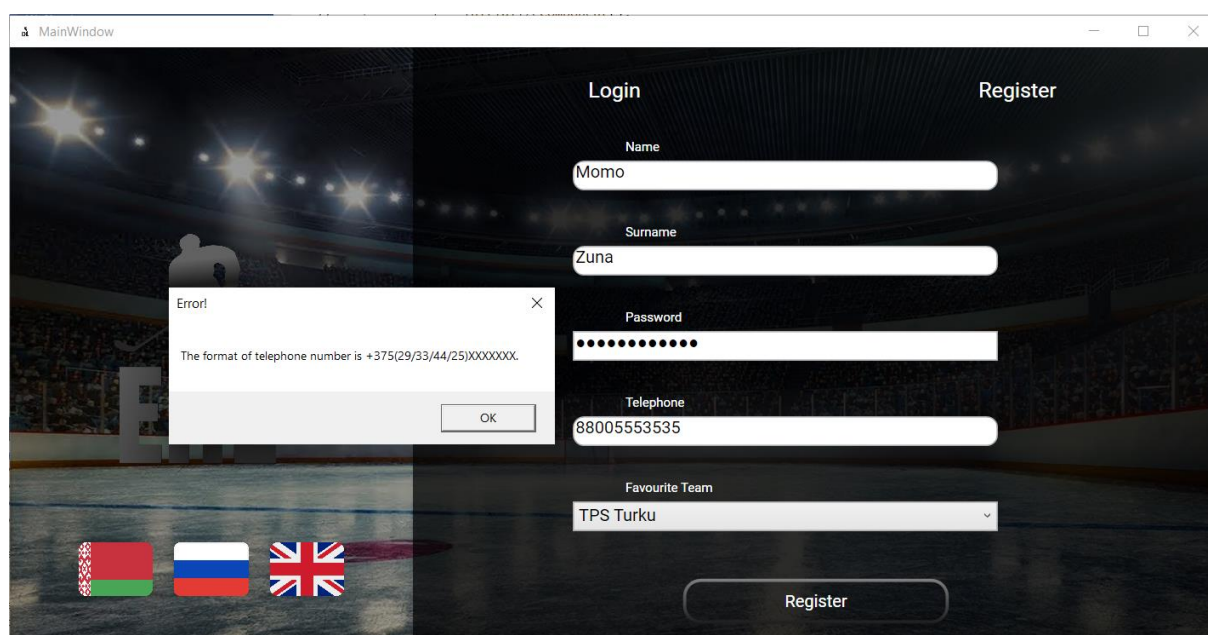


Рисунок 5.3 - обработка номера телефона неверного формата

В случае успешной регистрации пользователя перебрасывает на страницу с авторизацией, где ему необходимо ввести имя, фамилию и пароль. Проверяется наличие введенных имени и фамилии в базе данных, а также соответствие пароля. При успешном входе данное окно с регистрацией и авторизацией закрывается, и открывается окно с контентом.

Результаты обработки приведены на рисунках 5.4 и 5.5.

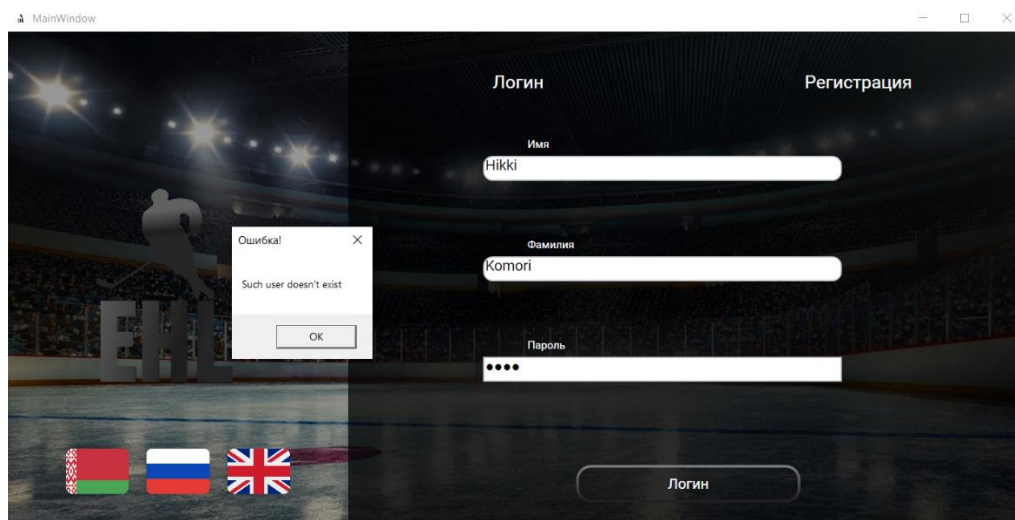


Рисунок 5.4 – обработка авторизации незарегистрированного пользователя

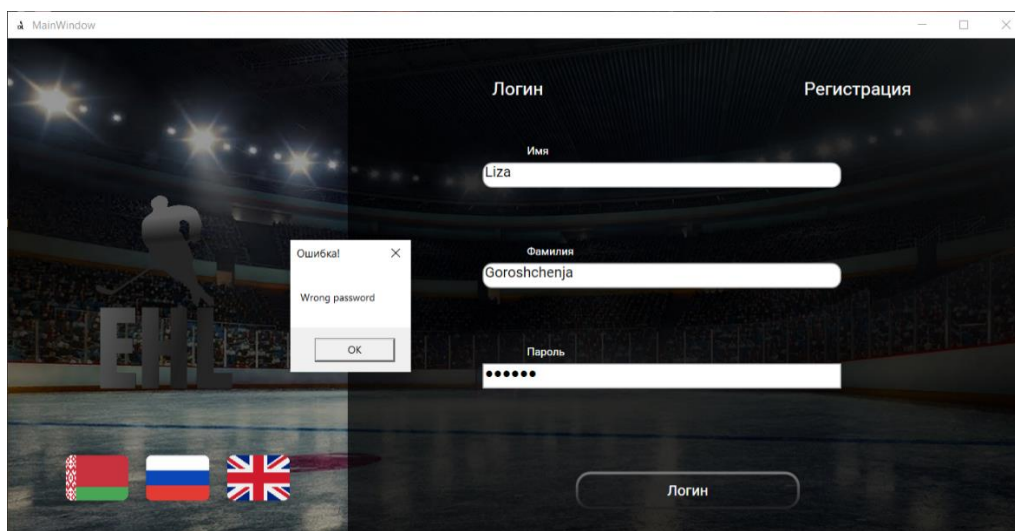


Рисунок 5.5 – обработка авторизации при вводе неверного пароля

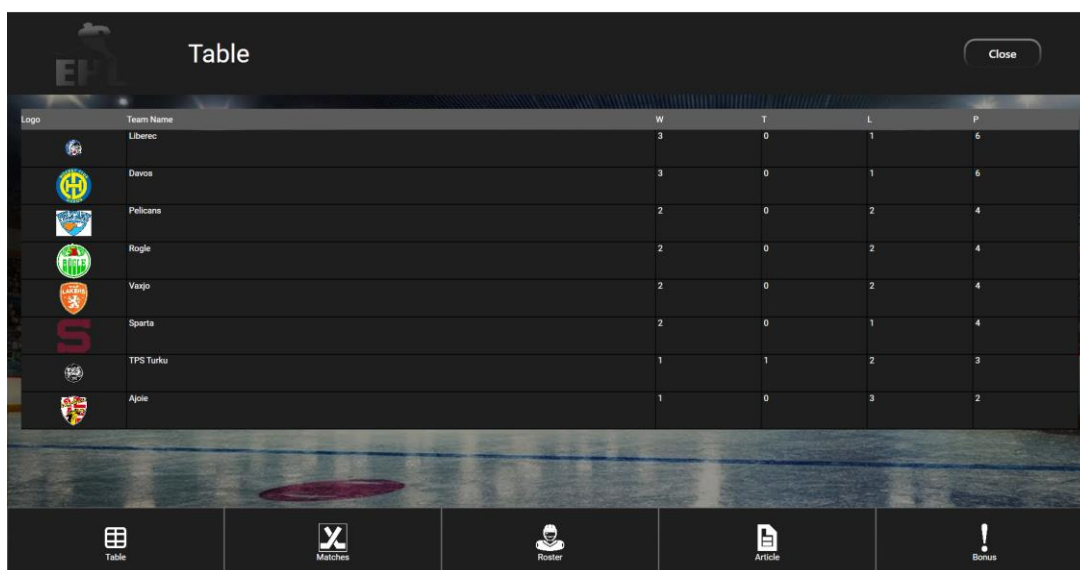
Таким образом, процесс регистрации проходит корректно и пользователь может зарегистрироваться в системе, а также войти в неё.

5.2 Тестирование изменения результата матчей

Изменение результата матчей – очень важный процесс работы приложения, влияющий на данные турнирной таблицы. Администратор обязан обновлять результат матча, и изменения, внесённые им, должны влиять на положение команды в общем зачёте. Исход матча имеет три случая: победа, ничья, поражение. За победу команде начисляется 2 очка, за ничью – 1, за поражение – 0. Количество очков есть сумма произведений числа каждого этого случая на коэффициент начисления. В основном принято сортировать команды по количеству очков, однако механизм

таблицы в Window Presentation Foundation позволяет сортировать данные по многим другим полям, таким как название команд, число побед, ничей и поражений.

Начальный сценарий для тестирования работоспособности обновления матчей представлен на рисунке 5.6.











Logo	Team Name	W	T	L	P
	Liberec	3	0	1	6
	Dava	3	0	1	6
	Pelicans	2	0	2	4
	Rogle	2	0	2	4
	Vaxjo	2	0	2	4
	Sparta	2	0	1	4
	TPS Turku	1	1	2	3
	Ajoie	1	0	3	2

Рисунок 5.6 – начальное положение команд в турнирной таблице

Перейдя на страницу с матчами, администратор выбирает нужный ему матч, в котором необходимо обновить результаты. После нажатия на кнопку обновления результатов, открывается специальное окно для изменения значений голов команды хозяев и гостей. Это проиллюстрировано на рисунке 5.7.

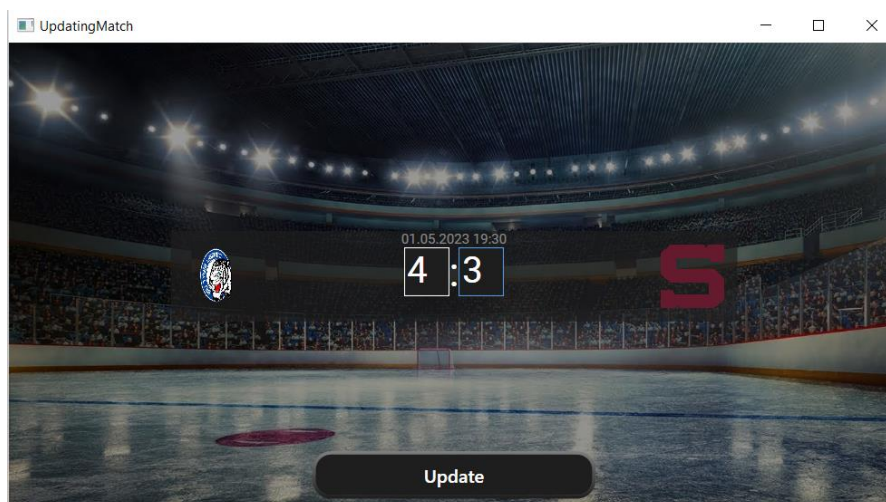
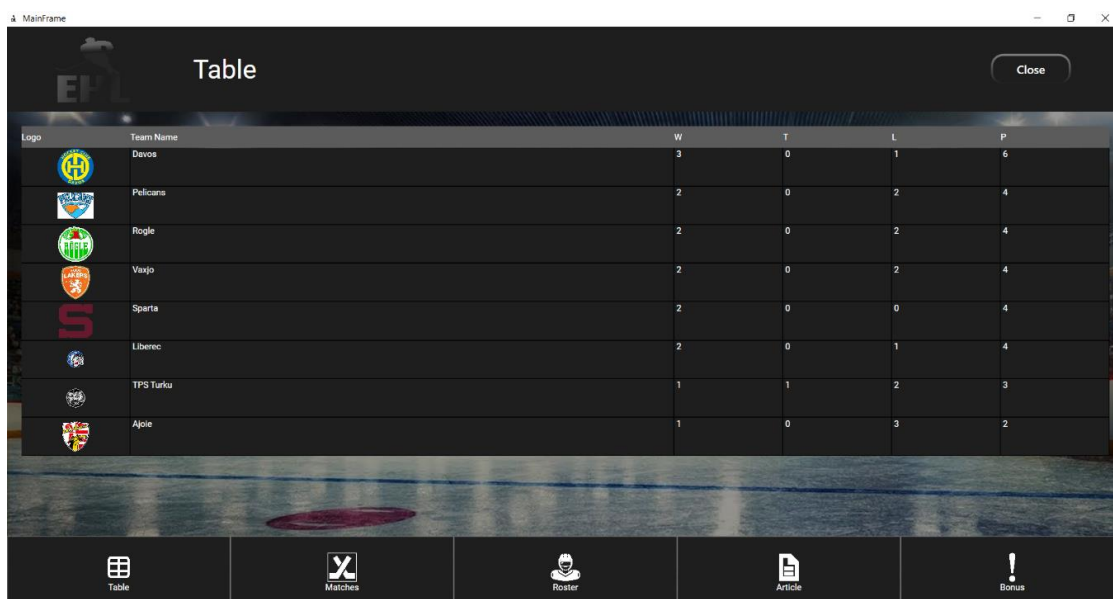


Рисунок 5.7 – окно для обновления результата матча

При этом, если матч до этого не обновлялся, то происходит откат результата. Откат результата подразумевает собой процесс уменьшения на единицу или числа побед, или поражений, или ничей. Очки команды также откатываются, но в

зависимости от предыдущего результата. Соответственно, происходят и изменения в турнирной таблице.

Обработка результата при обновлении матча проиллюстрирована на рисунке 5.8.



Logo	Team Name	W	T	L	P
	Devos	3	0	1	6
	Pelicans	2	0	2	4
	Rogle	2	0	2	4
	Vaxjo	2	0	2	4
	Sparta	2	0	0	4
	Liberec	2	0	1	4
	TPS Turku	1	1	2	3
	Alpie	1	0	3	2

Рисунок 5.8 – обработка отката результата при обновлении результата

В текстовых поля администратор должен ввести количество голов, которое забили одна и другая команды. В них должно содержаться целое значение больше 0. При вводе отрицательного, пустого, нецелого или значения, не являющегося числом, обновление результата не произойдёт, а выведется окно об ошибке. Это продемонстрировано на рисунке 5.9.

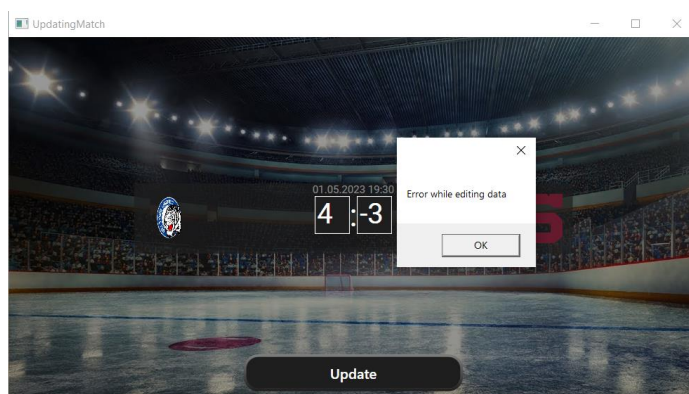


Рисунок 5.9 – обработка ввода отрицательного значения голов

При корректности введённых данных, данное окно закроется и выведется сообщение об успешном обновлении, в то же время произойдёт и обновление данных в турнирной таблице. Обновление данных матча и изменение положения в турнирной таблице приведено на рисунках 5.10 и 5.11.

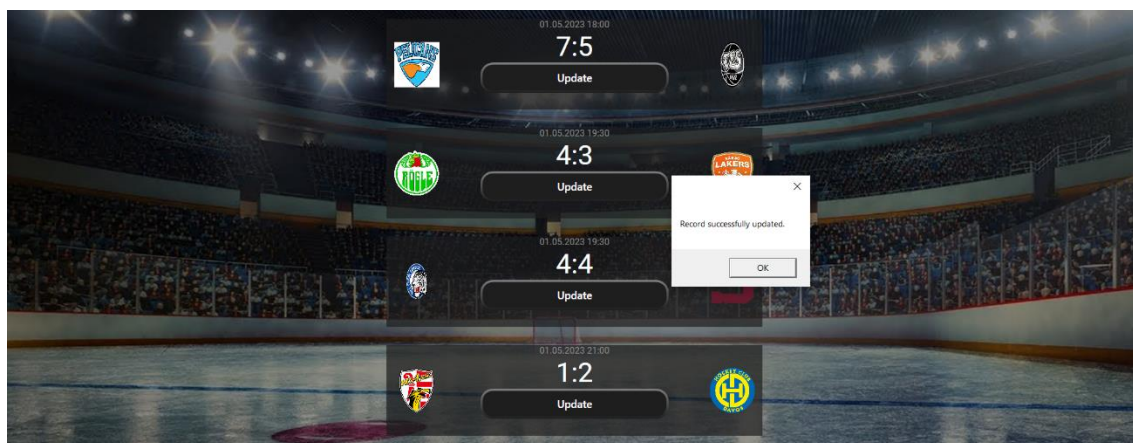


Рисунок 5.10 – сообщение об успешном обновлении результата матча

The screenshot shows a 'Table' window with a list of teams and their statistics. The table has columns for Logo, Team Name, W (Wins), T (Ties), L (Losses), and P (Points). The teams listed are Davos, Sparta, Liberec, Pelicans, Rogle, Vaxjo, TPS Turku, and Ajaccio.

Logo	Team Name	W	T	L	P
	Davos	3	0	1	6
	Sparta	2	1	0	5
	Liberec	2	1	1	5
	Pelicans	2	0	2	4
	Rogle	2	0	2	4
	Vaxjo	2	0	2	4
	TPS Turku	1	1	2	3
	Ajaccio	1	0	3	2

Рисунок 5.11 – изменение положения команд об общем зачёте

Таким образом, процесс обновления результатов матчей проходит корректно и процесс изменения ситуации в турнирной таблице происходит автоматически, что во многом упрощает работу администратора с подобными данными.

5.3 Тестирование изменений в составе команды

Любую хоккейную команду невозможно представить без наличия в ней игроков. Игроки в хоккее делятся на три типа: нападающие, защитники и вратари. Всего на площадке от команды участвуют 6: 3 нападающих, 2 защитника и вратарь. Согласно правилам, хоккейная команда должна иметь 4 звена, то есть 5 полевых игроков (вратарь не учитывается), но для тестового варианта можно использовать 6.

При открытии администратором страницы с составами, ему представляется список команд и, выбрав нужную ему команду, открывается специальное окно для работы с составами.

Специальное окно для работы с составами представлено на рисунке 5.12.

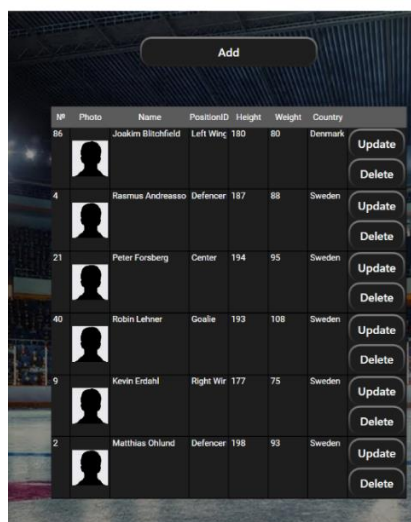


Рисунок 5.12 – окно администратора для работы с составом

По нажатию кнопки добавления или обновления, открывается специальное окно для заполнения данных игрока. При этом проверяется заполненность всех полей, корректность вводимых данных. Обработка пустых полей приведена на рисунке 5.13.

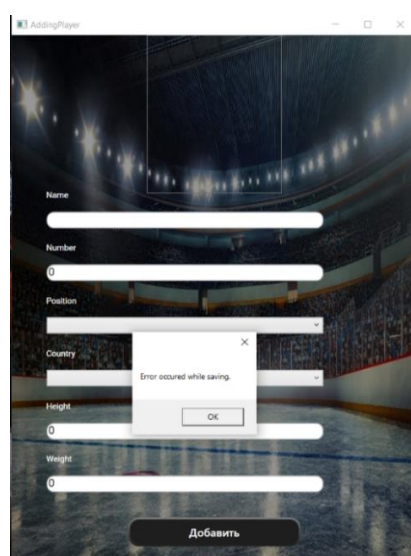


Рисунок 5.13 – обработка пустых полей

При заполненности всех полей, происходит проверка корректности данных. В частности, в составе команды не может находиться 2 игрока с одинаковым номером.

Обработка повторяющегося номера в составе приведена на рисунке 5.14.

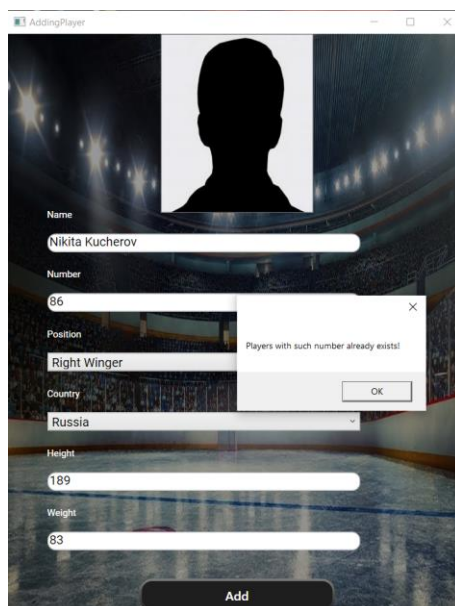


Рисунок 5.14 – обработка повторяющегося номера

Также проводится проверка на корректность ввода имени и фамилии, а также значения роста и веса. Имя и фамилия вводятся в одну строку латинскими буквами. Проверка корректности имени приведена на рисунке 5.15.

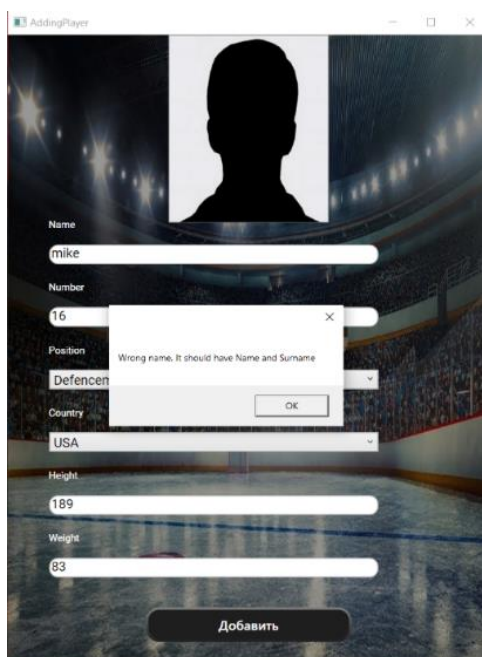


Рисунок 5.15 – проверка корректности имени

Значение роста может быть от 170 до 200, веса – от 65 до 110. Также должны быть заполнены поля с выбранной позицией игрока и выбранной страной.

Обработка некорректных значений роста и веса приведены на рисунках 5.16 и 5.17.

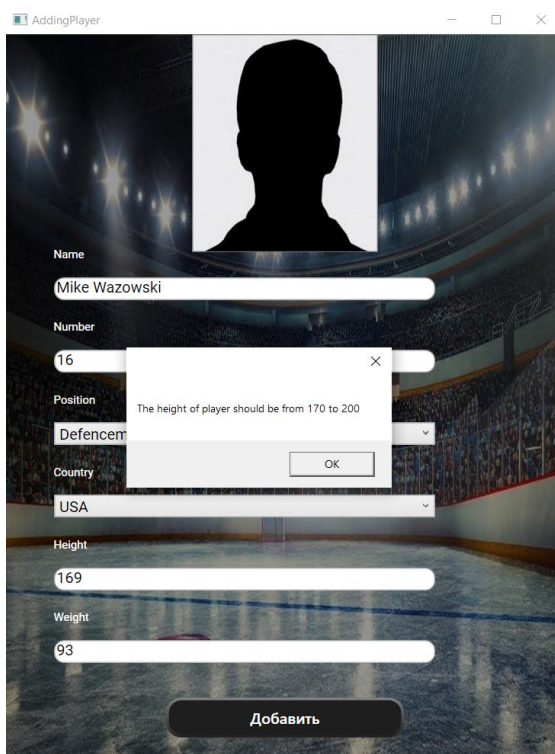


Рисунок 5.16 – обработка некорректного значения роста

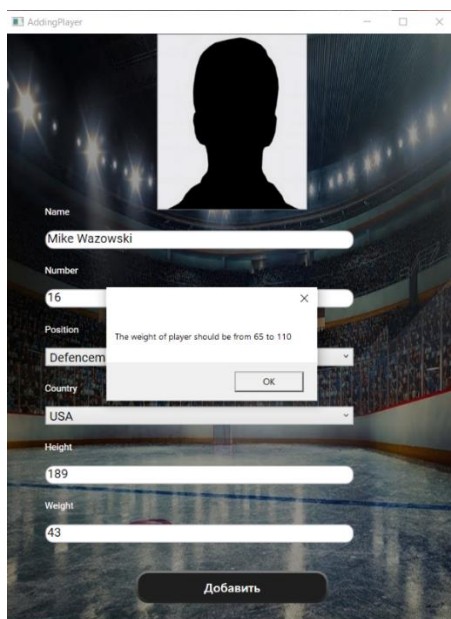


Рисунок 5.17 – обработка некорректного значения веса

Таким образом, процесс работы с игроками осуществляется корректно, и как администратор, так и пользователь могут отслеживать изменения состава каждой команды.

5.4 Тестирование добавления статьи и бонуса

Процесс добавления статьи и бонуса являются аналогичными процессами. На странице администратора имеется специальный элемент управления, включающий в себя добавление изображения и поле ввода текста. При их добавлении важно, чтобы ни одно поле не было пустым. Если все эти поля заполнены, то в список добавляются новая статья или же бонус. Результаты тестирования приведены на рисунках 5.18 – 5.22.

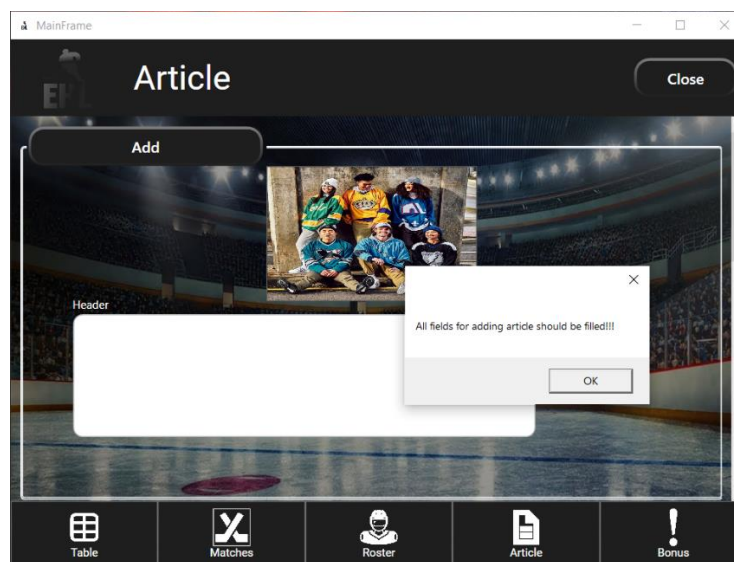


Рисунок 5.18 – обработка пустого заголовка

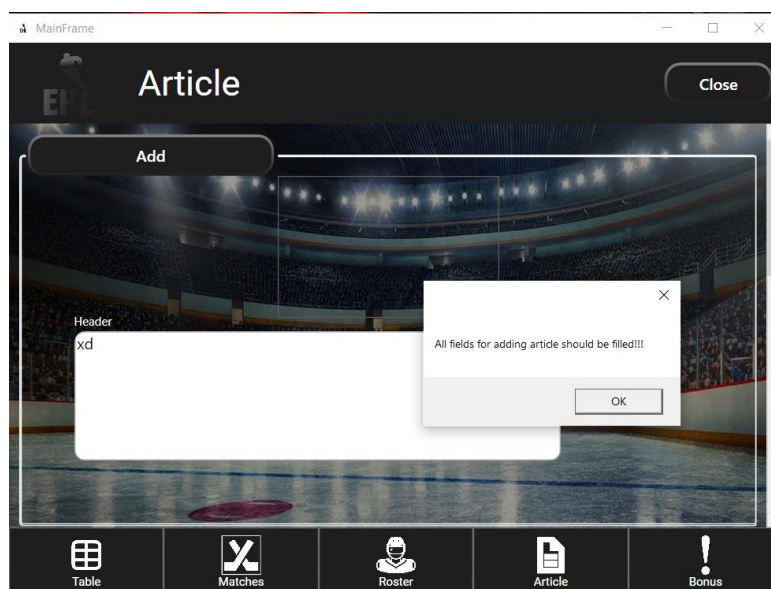


Рисунок 5.19 – обработка пустой фотографии

Таким образом, тестирование добавления новой статьи или бонуса является корректным, позволяя администратору добавлять их, а клиенту просматривать статьи и пользоваться бонусами.

6 Руководство по установке и использованию

6.1 Руководство по установке

Для работы с приложением понадобятся:

- Windows OS;
- MSSQL Server;
- .NET Framework 4.7.2.

Последовательность шагов при установке:

- клонировать репозиторий;
- настроить строку подключения «connectionString» пункт «DataSource» под свой SQL Server;
- запустить приложение.

6.2 Руководство по пользованию

При запуске приложения откроется окно для авторизации и регистрации. В нём также можно будет выбрать язык для всего интерфейса. При наличии аккаунта в приложении, пользователь переходит на страницу со входом, иначе регистрируется. Важно отметить, что роль администратора при регистрации не присваивается, то есть по умолчанию стоит роль пользователя. Страница регистрации на рисунке 6.1.

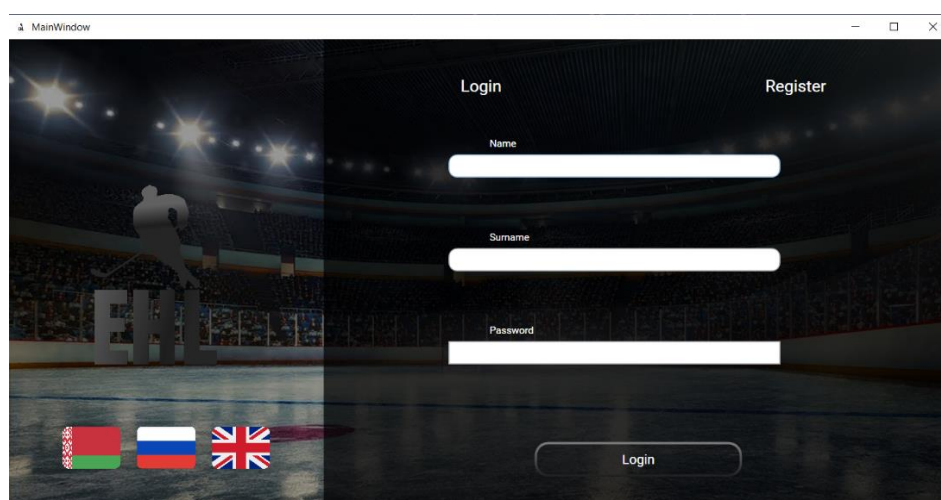


Рисунок 6.1 – страница регистрации

Если регистрация прошла успешно, то пользователя перебрасывает на страницу со входом. Проверяется соответствие введенных имени и фамилии с паролем, и в случае правильности, осуществляется переход на новое окно, а окно с авторизацией закрывается.

Страница регистрации представлена на рисунке 6.2.

The screenshot shows a window titled 'MainWindow' with a dark background featuring a hockey rink and stadium lights. On the left, there's a vertical banner with the EHL logo and flags of Belarus, Russia, and the UK. On the right, there's a 'Login' section with fields for Name, Surname, Password, and Telephone, and a 'Register' button. Below these is a 'Favourite Team' dropdown menu. At the bottom right, there's a 'Register' button.

Рисунок 6.2 – страница входа

После осуществления входа, открывается окно с контентом. В качестве главной страницы данного окна выступает турнирная таблица команд. Она приведена на рисунке 6.3.

The screenshot shows a window titled 'MainFrame' with a dark background. At the top, there's a 'Табліца' (Table) title and a 'Close' button. Below the title is a table with the following data:

Logo	Team Name	W	T	L	P
	Davos	3	0	1	6
	Sparta	2	1	0	5
	Liberec	2	1	1	5
	Pelicans	2	0	2	4
	Rogle	2	0	2	4
	Vaxjo	2	0	2	4
	TPS Turku	1	1	2	3

At the bottom of the window, there's a navigation bar with five icons and labels: 'Табліца' (Table), 'Матчы' (Matches), 'Ростары' (Rosters), 'Артыкулы' (Articles), and 'Бонус' (Bonus).

Рисунок 6.3 – окно с контентом

По нажатию кнопки с матчами на нижней панели осуществляется переход на страницу с результатами матчей, отсортированных по дате. Пользователь может прокручивать список матчей и смотреть результат.

Это представлено на рисунке 6.4.

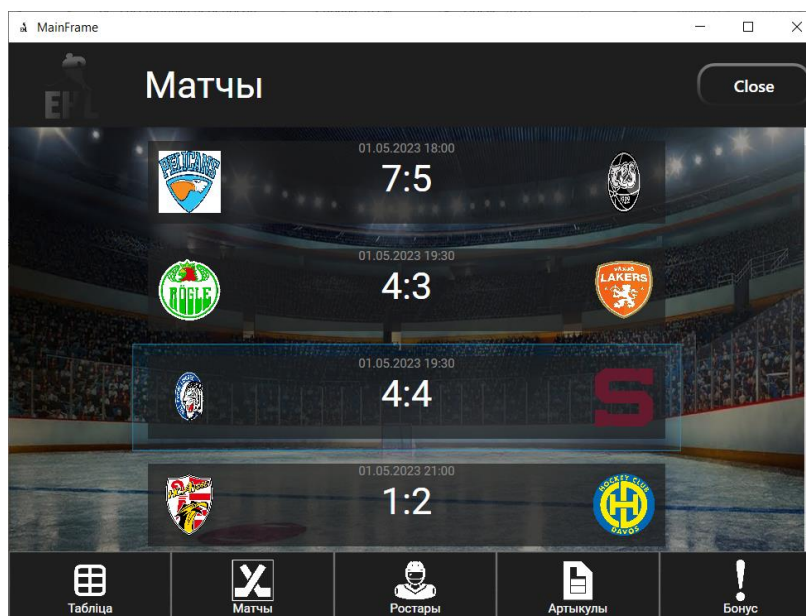


Рисунок 6.4 – страница с результатами матчей

По нажатию кнопки с составами на нижней панели появляется список команд, и выбрав необходимую, открывается окно с составом команды. Это представлено на рисунке 6.5.

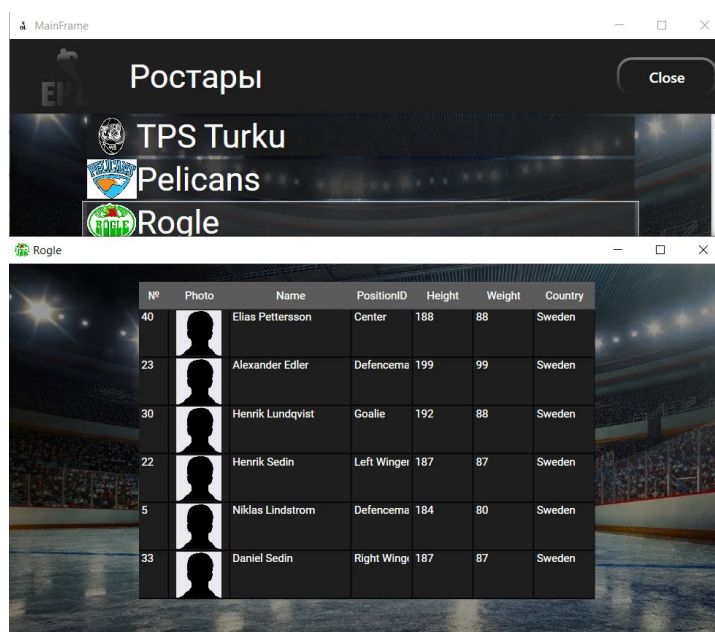


Рисунок 6.5 – страница для составов и окно с игроками

По нажатию кнопки со статьями на нижней панели пользователь может прокручивать заголовки, тем самым, быть уведомлённым о последних событиях в мире лиги.

Страница со статьями представлена на рисунке 6.6.

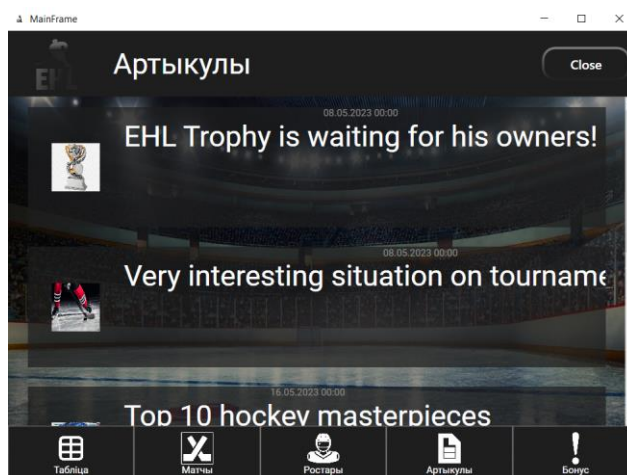


Рисунок 6.6 – страница со статьями

По нажатию кнопки с бонусами на нижней панели открывается список доступных бонусов, и выбрав понравившийся, появится окошко с просьбой ожидания обратной связи с администратором.

Страница с бонусами представлена на рисунках 6.7.

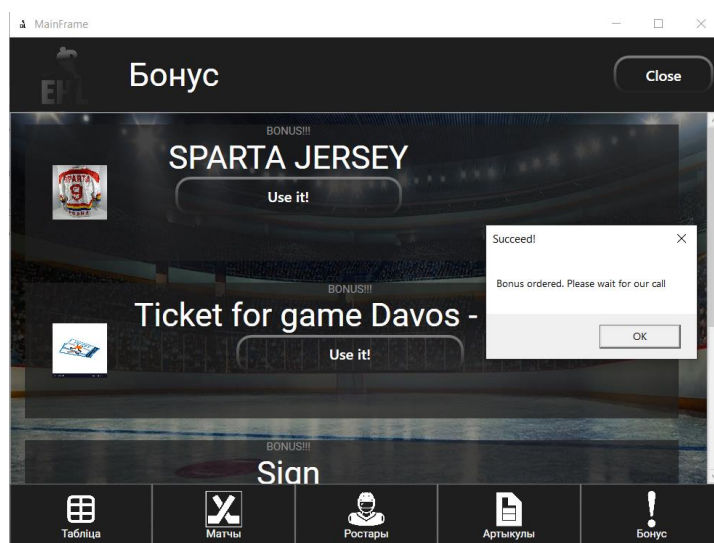


Рисунок 6.7 – страница с бонусами

После выбора бонуса пользователем, администратору приходит сообщение на телефон, где указано имя, фамилия пользователя, его номер телефона и название бонуса. Также администратор может посмотреть это в специальном окне на своей панели, где содержится уже более подробная информация.

Заключение

В заключение хочется отметить, что в ходе работы было разработано приложение для турнирной таблицы турнира Enjoy Hockey League. Приложения данного типа встречаются сейчас очень часто, что способствует популяризации хоккея как в стране, так и в мире.

Для разработки приложения использовались следующие технологии: .NET Framework 4.7.2, MSSQL Server, Window Presentation Foundations, Entity Framework Core. Они позволили реализовать широкий спектр задач, таких как авторизация, аутентификация и регистрация пользователей, обновление результатов матчей, изменение составов команд и возможность пользования бонусом.

Были реализованы такие паттерны проектирования как Repository и UnitOfWork. Данные паттерны очень широко используются при разработке программного обеспечения, позволяя упростить работу с изменяющимися данными и доступ к ним.

В ходе работы были выполнен список функциональных требований, приведённый ниже:

Для пользователя:

- возможность регистрации;
- возможность авторизации;
- возможность выбора любимой команды;
- просмотр турнирной таблицы;
- просмотр результатов матчей;
- просмотр составов команд;
- пользование бонусом;
- возможность смены языка и темы;
- просмотр статей;
- сортировка данных в таблице;

Для администратора:

- возможность авторизации;
- просмотр и обновление турнирной таблицы;
- просмотр и обновление результатов матчей;
- просмотр составов команд;
- предоставление бонуса клиенту;
- просмотр и добавление статей;
- возможность поддержки связи с пользователем;

В ходе тестирования было выявлено, что приложение работает корректно, выполняет все свои функции и обрабатывает все возможные негативные сценарии.

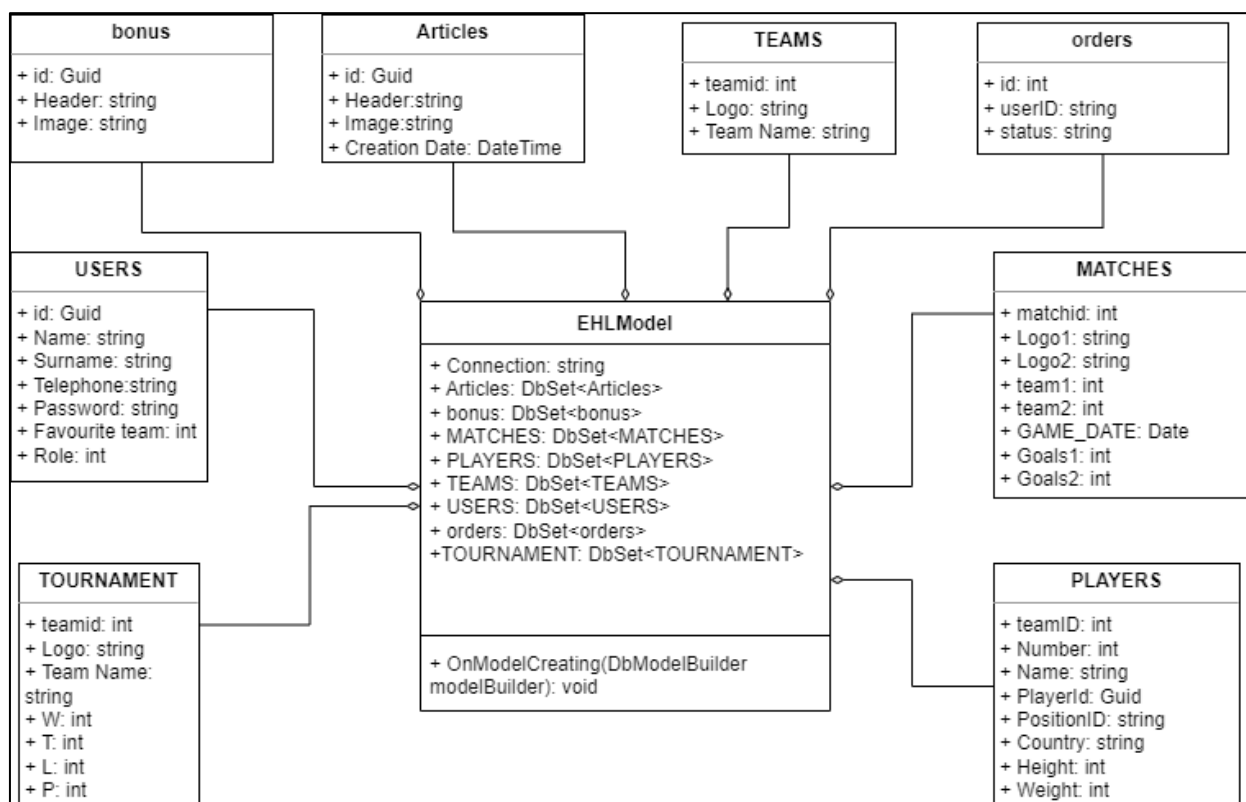
Также приведено руководство по установке и пользованию, содержащее подробные инструкции для работы с приложением. В соответствии с полученным результатом можно сделать вывод, что разработанный программный продукт выполнен верно с соблюдением требований технического задания.

Список использованных источников

1. Руководство по WPF [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/wpf/#google_vignette – Дата доступа: 24.04.2023.
2. Руководство по Entity Framework [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/efcore/> – Дата доступа: 28.04.2023.
3. WPF CRUD Application Using DataGrid, MVVM Pattern, Entity Framework, And C#.NET [Электронный ресурс]: <https://dotnetgenetics.blogspot.com/2021/02/wpf-crud-with-datagrid-mvvm-entity.html> – Дата доступа: 25.04.2023
4. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
5. Блинова, Е.А. Курс лекций по Бадам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

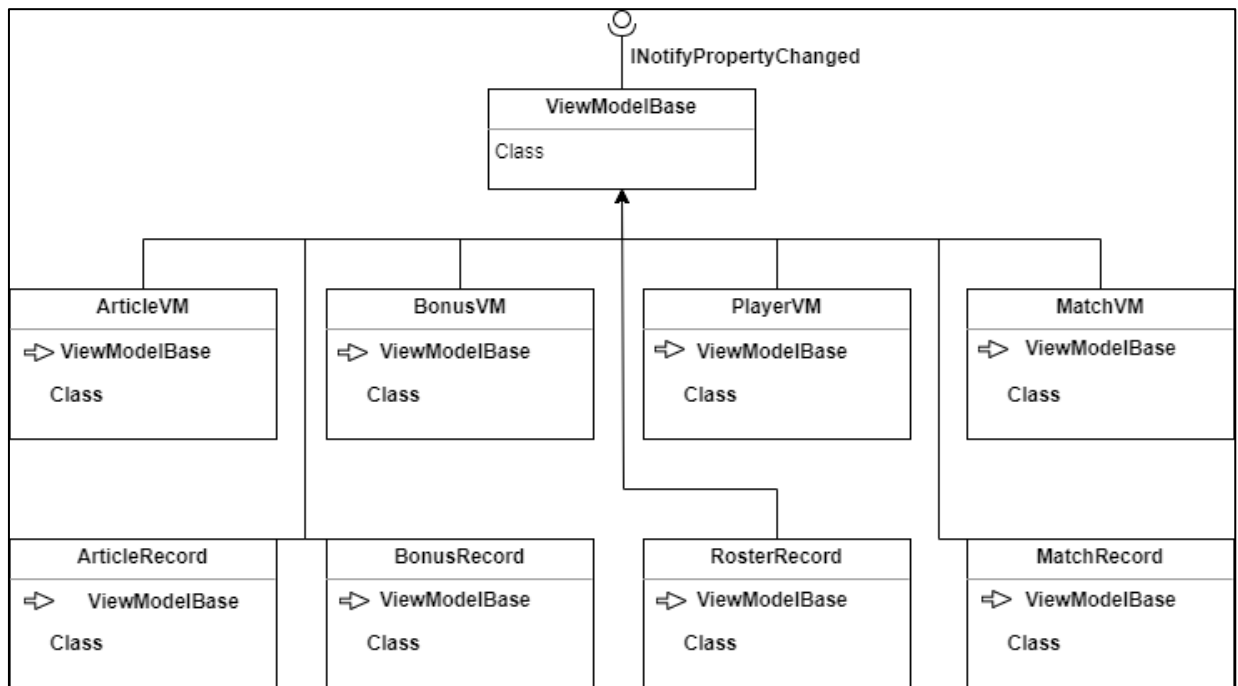
ПРИЛОЖЕНИЕ А

Взаимодействие классов модели



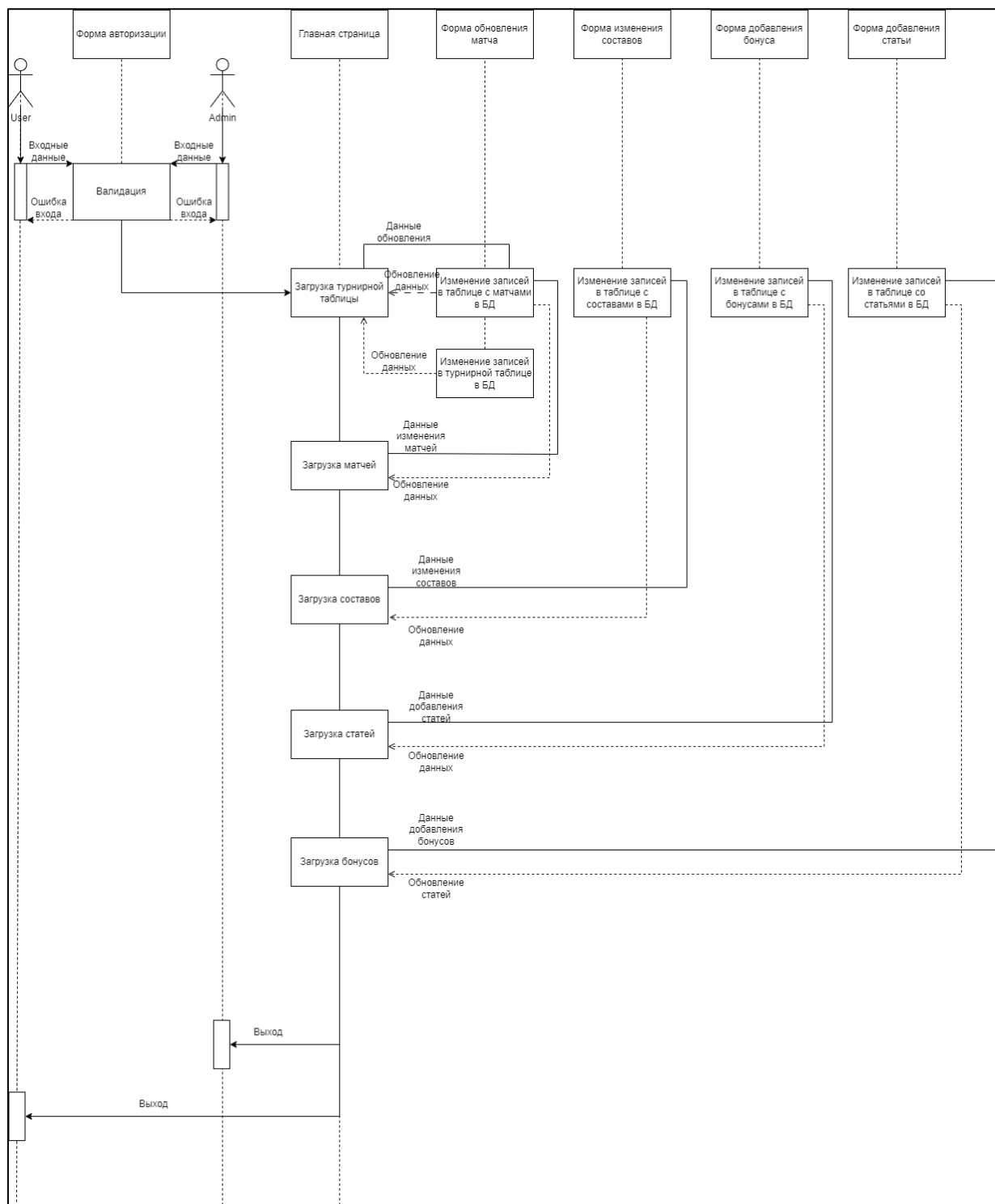
ПРИЛОЖЕНИЕ Б

Взаимодействие классов модели представления



ПРИЛОЖЕНИЕ В

Диаграмма последовательностей



ПРИЛОЖЕНИЕ Г

Реализация паттерна UnitOfWork

```
public class UnitOfWork : IDisposable
{
    private EHLModel context = new EHLModel();
    private bool disposed = false;

    private UserRepo usersRepo;
    private MatchesRepo matchesRepo;
    private PlayersRepo playersRepo;
    private ArticlesRepo articlesRepo;
    private BonusRepo bonusRepo;
    private OrdersRepo ordersRepo;

    public BonusRepo BonusRepo
    {
        get
        {
            if (bonusRepo == null)
                bonusRepo = new BonusRepo();
            return bonusRepo;
        }
    }

    public ArticlesRepo ArticlesRepo
    {
        get
        {
            if (articlesRepo == null)
                articlesRepo = new ArticlesRepo();
            return articlesRepo;
        }
    }

    public MatchesRepo MatchesRepo
    {
        get
        {
            if (matchesRepo == null)
                matchesRepo = new MatchesRepo();
            return matchesRepo;
        }
    }
}
```



```

public PlayersRepo PlayersRepo
{
    get
    {
        if (playersRepo == null)
            playersRepo = new PlayersRepo();
        return playersRepo;
    }
}

public UserRepo UsersRepo
{
    get
    {
        if (usersRepo == null)
            usersRepo = new UserRepo();
        return usersRepo;
    }
}

public OrdersRepo OrdersRepo
{
    get
    {
        if (ordersRepo == null)
            ordersRepo = new OrdersRepo();
        return ordersRepo;
    }
}

public void Save() => context.SaveChanges();

public virtual void Dispose(bool disposing)
{
    if (!this.disposed)
    {
        if (disposing)
            context.Dispose();
        this.disposed = true;
    }
}

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
}

```

ПРИЛОЖЕНИЕ Д

Листинг 1 – реализация команды для изменения результата матча

```
private ICommand _editCommand;
public ICommand EditCommand
{
    get
    {
        if (_editCommand == null)
            _editCommand = new RelayCommand(param =>
EditData((int)param), null);

        return _editCommand;
    }
}
private MatchesRepo _repository = new MatchesRepo();
public void EditData(int id)
{
    try
    {
        var model = _repository.Get(id);
        this.matchid = model.matchid;
        this.Goals1 = (int)model.Goals1;
        this.Goals2 = (int)model.Goals2;
        this.Logo1 = model.Logo1;
        this.Logo2 = model.Logo2;
        this.GAME_DATE = (DateTime)model.GAME_DATE;
        var window = new UpdatingMatch(id);
        EHLModel ehl = new EHLModel();
        window.DataContext = this;
        window.Show();
        var match = new SqlParameter("@id", id);
        if (Goals1 > Goals2 && (Goals1 >= 0 && Goals2 >= 0))
        {
            ehl.Database.ExecuteNonQuery("update
tournament set W=W-1,P=P-2 from MATCHES where teamid=MATCHES.Team1
and matchid=@id " +
                "update tournament set L = L - 1 from
MATCHES where teamid = MATCHES.Team2 and matchid = @id", match);
        }
        else if (Goals1 == Goals2 && (Goals1 >= 0 && Goals2
>= 0))
        {
            if (Goals1 != 0 && Goals2 != 0)
            {
                ehl.Database.ExecuteNonQuery("update
tournament set T = T - 1, P = P - 1 from MATCHES where (teamid =
MATCHES.Team1 or teamid = MATCHES.Team2) and matchid = @id", match);
            }
        }
    }
}
```

```

else if (Goals1 < Goals2 && (Goals1 >= 0 && Goals2
>= 0))
{
    ehl.Database.ExecuteNonQuery("update
tournament set W=W-1,P=P-2 from MATCHES where teamid=MATCHES.Team2
and matchid=@id " +
                                "update tournament set L = L - 1 from MATCHES
where teamid = MATCHES.Team1 and matchid = @id", match);
}
if (Goals1 < 0 && Goals2 < 0)
{
    throw new Exception();
}
}
catch(Exception e)
{
    MessageBox.Show("Wrong input data... You can only
input integers that bigger than 0");
}}

```

Листинг 2 – реализация команды для удаления и обновления игрока

```
private ICommand _editCommand;
private ICommand _deleteCommand;
public RosterRecord GetRecord(Guid id)
{
    return PlayersRecords.Where(x => x.PlayerId ==
id).FirstOrDefault();
}
public ICommand EditCommand
{
    get
    {
        if (_editCommand == null)
            _editCommand = new RelayCommand(param =>
EditData((Guid)param), null);

        return _editCommand;
    }
}

public ICommand DeleteCommand
{
    get
    {
        if (_deleteCommand == null)
            _deleteCommand = new RelayCommand(param =>
DeletePlayer((Guid)param), null);

        return _deleteCommand;
    }
}

public void DeletePlayer(Guid id)
{
    if (MessageBox.Show("Confirm delete of this record?",
"Player", MessageBoxButton.YesNo)
    == MessageBoxResult.Yes)
    {
        try
        {
            _repository.RemovePlayer(id);
            MessageBox.Show("Record successfully deleted.");
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error occured while saving. " +
ex.InnerException);
        }
        finally
        {

```

```

        foreach (Window window in
Application.Current.Windows)
        {
            if (window.GetType() ==
typeof(AdminLookRosters))
            {
                (window as
AdminLookRosters).table.Items.Refresh();
                (window as
AdminLookRosters).table.ItemsSource = (new
PlayerVM()).EHLRecord.PlayersRecords.Where(x => x.teamID ==
this.teamID);
            }
        }
        _repository.GetAll();
    }
}

public void EditData(Guid data)
{
    var model = _repository.Get(data);
    this.teamID = (int)model.teamID;
    this.Number = (int)model.Number;
    this.ImagePath = model.Photo;
    this.Photo = model.Photo;
    this.Name = model.Name;
    this.PositionID = model.PositionID;
    this.Country = model.Country;
    this.PlayerId = model.PlayerId;
    this.Height = model.Height;
    this.Weight = model.Weight;
    var window = new UpdatingPlayer(this.teamID);
    window.DataContext = this;
    window.Show();
}

private ICommand _saveCommand;
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
            _saveCommand = new RelayCommand(param =>
SaveData(), null);

        return _saveCommand;
    }
}

public void SaveData()
{
    if (this != null)
    {

```

```

        playerEntity.teamID = this.teamID;
        playerEntity.Photo = this.ImagePath;
        playerEntity.Number = this.Number;
        playerEntity.Name = this.Name;
        playerEntity.PlayerId = this.PlayerId;
        playerEntity.PositionID = this.PositionID;
        playerEntity.Country = this.Country;
        playerEntity.Weight = this.Weight;
        playerEntity.Height = this.Height;
        try
        {
            _repository.RemovePlayer(PlayerId);
            if (_repository.AddPlayer(playerEntity.teamID,
playerEntity.Number, playerEntity.Photo,
                playerEntity.Name, playerEntity.PositionID,
playerEntity.Country, playerEntity.PlayerId,
                playerEntity.Height, playerEntity.Weight))
            {
                _repository.UpdatePlayer(playerEntity);
                MessageBox.Show("Record successfully
updated.");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error occured while saving. " +
ex.InnerException);
        }
        finally
        {
            foreach (Window window in
Application.Current.Windows)
            {
                if (window.GetType() ==
typeof(AdminLookRosters))
                {
                    (window as
AdminLookRosters).table.Items.Refresh();
                    (window as
AdminLookRosters).table.ItemsSource = (new
PlayerVM()).EHLRecord.PlayersRecords.Where(x => x.teamID ==
playerEntity.teamID);
                }
            }
            _repository.GetAll();
        }
    }
    public void GetAll()
    {

```

```
        this.PlayersRecords = new
ObservableCollection<RosterRecord>();
        _repository.GetAll().ForEach(data =>
this.PlayersRecords.Add(new RosterRecord()
{
    teamID = data.teamID,
    Number = Convert.ToInt32(data.Number),
    Photo = data.Photo,
    Name = data.Name,
    PositionID = data.PositionID,
    PlayerId = data.PlayerId,
    Country = data.Country,
    Height = data.Height,
    Weight = data.Weight
}));
}
```


Листинг 3 – команда для добавления игрока в состав и сохранения изменений

```
public ICommand AddCommand
{
    get
    {
        if (_addCommand == null)
            _addCommand = new RelayCommand(param =>
AddData((int)param), null);

        return _addCommand;
    }
}
public void AddData(int id)
{
    PlayerVM rr = new PlayerVM();
    var window = new AddingPlayer(id);
    window.DataContext = rr;
    rr.EHLRecord.teamID = id;
    window.Show();
}
private ICommand _saveCommand;
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
            _saveCommand = new RelayCommand(param =>
SaveData(), null);

        return _saveCommand;
    }
}
public void SaveData()
{
    if (this != null)
    {
        playerEntity.teamID = EHLRecord.teamID;
        playerEntity.Photo = EHLRecord.ImagePath;
        playerEntity.Number = EHLRecord.Number;
        playerEntity.Name = EHLRecord.Name;
        playerEntity.PositionID = EHLRecord.PositionID;
        playerEntity.PlayerId = Guid.NewGuid();
        playerEntity.Country = EHLRecord.Country;
        playerEntity.Weight = EHLRecord.Weight;
        playerEntity.Height = EHLRecord.Height;
        try
        {
            if (_repository.AddPlayer(playerEntity.teamID,
playerEntity.Number, playerEntity.Photo,
```

```

        playerEntity.Name, playerEntity.PositionID,
playerEntity.Country, playerEntity.PlayerId,
        playerEntity.Height, playerEntity.Weight))
    {
        _repository.UpdatePlayer(playerEntity);
        MessageBox.Show("Record successfully
updated.");
    }
    else throw new Exception();
}
catch (Exception ex)
{
    MessageBox.Show("Error occured while saving. " +
ex.InnerException);
}
finally
{
    foreach (Window window in
Application.Current.Windows)
    {
        if (window.GetType() ==
typeof(AdminLookRosters))
        {
            (window as
AdminLookRosters).table.Items.Refresh();
            (window as
AdminLookRosters).table.ItemsSource = (new
PlayerVM()).EHLRecord.PlayersRecords.Where(x => x.teamID ==
playerEntity.teamID);
        }
    }
    //_repository.GetAll();
}
}
}

```

Листинг 4 – команда для добавления статьи

```
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
            _saveCommand = new RelayCommand(param =>
SaveData(), null);

        return _saveCommand;
    }
}
public ArticleVM()
{
    AEntity = new Articles();
    _repository = new ArticlesRepo();
    ARecord = new ArticleRecord();
    GetAll();
}

public void SaveData()
{
    if (ARecord != null)
    {
        AEntity.id = Guid.NewGuid();
        AEntity.Image = ARecord.ImagePath;
        AEntity.Header = ARecord.Header;
        AEntity.Creation_Date = (DateTime)DateTime.Now;
        try
        {
            if (AEntity.Image != null && AEntity.Header
!= null)
            {
                _repository.AddArticle(AEntity);
                MessageBox.Show("New record
successfully saved.");
            }
            else throw new Exception();
        }
        catch (Exception ex)
        {
            MessageBox.Show("All fields for adding
article should be filled!!! " + ex.InnerException);
        }
        finally
        {
            GetAll();
        }
    }
}
```

Листинг 5 – команда для добавления бонуса

```
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
            _saveCommand = new RelayCommand(param =>
SaveData(), null);

        return _saveCommand;
    }
}
public BonusVM()
{
    BEntity = new bonus();
    _repository = new BonusRepo();
    BRecord = new BonusRecord();
    GetAll();
}

public void SaveData()
{
    if (BRecord != null)
    {
        BEntity.id = Guid.NewGuid();
        BEntity.image = BRecord.ImagePath;
        BEntity.header = BRecord.Header;
        try
        {
            if (BEntity.image != null &&
String.IsNullOrEmpty(BEntity.header))
            {
                _repository.AddBonus(BEntity);
                MessageBox.Show("New record
successfully saved.");
            }
            else
            {
                throw new Exception();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error occurred while
saving. All fields for adding bonus should be filled!" +
ex.InnerException);
        }
        finally
        {

```

```
        GetAll();  
    }  
}
```

ПРИЛОЖЕНИЕ Е

Реализация класса Registration

```
public partial class Registration : Page
{
    private UnitOfWork context;
    public Registration()
    {
        InitializeComponent();
        context = new UnitOfWork();
        FillCombobox();
    }

    private void login_Click(object sender, RoutedEventArgs e)
    {
        NavigationService.Navigate(new Uri("./Login.xaml",
UriKind.Relative));
    }

    private void MoveToMainFrame_Click(object sender,
RoutedEventArgs e)
    {
        var regex = new
Regex(@"^((\+375)((29)|(33)|(44)|(25))\d{7})$");
        if (string.IsNullOrEmpty(namebox.Text) ||
string.IsNullOrEmpty(surnamebox.Text) ||
string.IsNullOrEmpty(SecureStringToString(passwordbox.SecurePassword
)) || string.IsNullOrEmpty(favteam.Text))
            MessageBox.Show("Please, fill all fields", "Error!",
MessageBoxButton.OK);
        else if (!regex.IsMatch(telephonebox.Text))
            MessageBox.Show("The format of telephone number is
+375(29/33/44/25)XXXXXX.", "Error!", MessageBoxButton.OK);
        else
        {
            try
            {
                var result =
context.UsersRepo.AddNewUser(Guid.NewGuid(), namebox.Text,
surnamebox.Text,
SecureStringToString(passwordbox.SecurePassword), telephonebox.Text,
favteam.SelectedIndex + 1);
                if (result)
                {
                    MessageBox.Show("You succesfully
registered!", "Success!", MessageBoxButton.OK);
                    NavigationService.Navigate((new
Uri("./Login.xaml", UriKind.Relative)));
                }
            }
            catch { }
        }
    }
}
```

```

        }
        else throw new Exception();
    }
    catch
    {
        MessageBox.Show("Already exist", "Error!",
        MessageBoxButtons.OK);
    }
}

private void FillCombobox()
{
    string ConString =
    ConfigurationManager.ConnectionStrings["EHLModel"].ConnectionString;
    using (SqlConnection con = new SqlConnection(ConString))
    {
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "SELECT teamid,[Team Name] from
TEAMS";

        cmd.Connection = con;
        SqlDataAdapter sda = new SqlDataAdapter(cmd);
        DataTable dt = new DataTable("TOURNAMENT");
        sda.Fill(dt);
        favteam.ItemsSource = dt.DefaultView;
        favteam.DisplayMemberPath = "[Team Name]";
    }
}

private static string SecureStringToString(SecureString
value)
{
    IntPtr valuePtr = IntPtr.Zero;
    try
    {
        valuePtr =
        Marshal.SecureStringToGlobalAllocUnicode(value);
        return Marshal.PtrToStringUni(valuePtr);
    }
    finally
    {
        Marshal.ZeroFreeGlobalAllocUnicode(valuePtr);
    }
}
}

```


ПРИЛОЖЕНИЕ Ж

Реализация класса Login

```
public partial class Login : Page
{
    private UnitOfWork context;

    public Login()
    {
        InitializeComponent();
        context = new UnitOfWork();
    }
    private void reg_Click(object sender, RoutedEventArgs e)
    {
        NavigationService.Navigate(new
Uri("./Registration.xaml", UriKind.Relative));
    }
    private void MoveToMainFrame_Click(object sender,
RoutedEventArgs e)
    {
        var name = namebox.Text;
        var surname = surnamebox.Text;
        var password =
SecureStringToString(passwordbox.Password);
        if (string.IsNullOrEmpty(name))
        {
            MessageBox.Show("Enter name!", "Ошибка!",
MessageBoxButton.OK);
        }
        else if (string.IsNullOrEmpty(surname))
        {
            MessageBox.Show("Enter surname!", "Ошибка!",
MessageBoxButton.OK);
        }
        else if (string.IsNullOrEmpty(password))
        {
            MessageBox.Show("Enter password!", "Ошибка!",
MessageBoxButton.OK);
        }
        else
        {
            var user =
context.UsersRepo.GetUserByNameAndSurname(name, surname);
            if (user == null)
            {
                MessageBox.Show("Such user doesn't exist",
"Ошибка!", MessageBoxButton.OK);
            }
            else if (user.PASSWORD != password)
```

```

        MessageBox.Show("Wrong password", "Ошибка!",
MessageBoxButton.OK);
        else
        {
            try
            {
                MainFrame mainFrame = new MainFrame(user);
                mainFrame.Show();
                Window.GetWindow(this).Close();
            }
            catch
            {
                MessageBox.Show("Error. Please, try again
later", "Ошибка!", MessageBoxButton.OK);
            }
        }
    }
}

private static string SecureStringToString(SecureString
value)
{
    IntPtr valuePtr = IntPtr.Zero;
    try
    {
        valuePtr =
Marshal.SecureStringToGlobalAllocUnicode(value);
        return Marshal.PtrToStringUni(valuePtr);
    }
    finally
    {
        Marshal.ZeroFreeGlobalAllocUnicode(valuePtr);
    }
}
}

```

ПРИЛОЖЕНИЕ 3

Реализация класса Matches

```
public partial class Matches : Page
{
    public List<MATCHES> matches { get; set; }
    public Matches()
    {
        InitializeComponent();
        FillData();
    }
    public void FillData()
    {
        string ConString =
ConfigurationManager.ConnectionStrings["EHLModel"].ConnectionString;
        using (SqlConnection con = new SqlConnection(ConString))
        {
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "SELECT
Logo1,Goals1,Goals2,Logo2,GAME_DATE FROM Matches order by
GAME_DATE";

            cmd.Connection = con;
            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataTable dt = new DataTable("Matches");
            sda.Fill(dt);
            MatchList.ItemsSource = dt.DefaultView;
        }
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        foreach (Window window in Application.Current.Windows)
        {
            if (window.GetType() == typeof(MainFrame))
            {
                var wind = new MainWindow();
                wind.frameAuth.Content = new Login();
                wind.Show();
                window.Close();
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ И

Реализация класса Bonus

```
public partial class Bonus : Page
{
    public UnitOfWork unitOfWork;
    public USERS user;
    public bonus bonuS;
    public EHLModel ehl = new EHLModel();
    public Bonus(USERS userSignedIn)
    {
        InitializeComponent();
        DataContext = new BonusVM();
        unitOfWork = new UnitOfWork();
        user = userSignedIn;
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        foreach (Window window in Application.Current.Windows)
        {
            if (window.GetType() == typeof(MainFrame))
            {
                var wind = new MainWindow();
                wind.frameAuth.Content = new Login();
                wind.Show();
                window.Close();
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ К

Реализация класса AdminMatches

```
public partial class AdminMatches : Page
{
    private UnitOfWork context;
    public AdminMatches()
    {
        InitializeComponent();
        context = new UnitOfWork();
        this.DataContext = new MatchVM();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        foreach (Window window in Application.Current.Windows)
        {
            if (window.GetType() == typeof(MainFrame))
            {
                var wind = new MainWindow();
                wind.frameAuth.Content = new Login();
                wind.Show();
                window.Close();
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Л

Листинг 1 - Реализация класса AdminRosters

```
public partial class AdminRosters : Page
{
    private EHLModel model;
    public AdminRosters()
    {
        InitializeComponent();
        model = new EHLModel();
        FillExpanders();
    }
    public void FillExpanders()
    {
        string ConString =
ConfigurationManager.ConnectionStrings["EHLModel"].ConnectionString;
        using (SqlConnection con = new SqlConnection(ConString))
        {
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "SELECT TEAMS.teamid, TEAMS.Logo,
TEAMS.[Team Name] from TEAMS";
            cmd.Connection = con;
            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataTable dt = new DataTable("EXPANDERA");
            sda.Fill(dt);
            ListViewTeams.ItemsSource = dt.DefaultView;
        }
    }
    private void ListViewItem_PreviewMouseLeftButtonDown(object
sender, MouseButtonEventArgs e)
    {
        var item = sender as ListViewItem;
        if (item != null && item.IsSelected)
        {
            int i = ListViewTeams.SelectedIndex + 1;
            var window = new AdminLookRosters(i);
            var title = model.TEAMS.Where(n => n.teamid ==
ListViewTeams.SelectedIndex + 1).Single();
            window.Title = title.Team_Name;
            window.Icon =
BitmapFrame.Create(Application.GetResourceStream(new Uri(title.Logo,
UriKind.RelativeOrAbsolute)).Stream);
            window.Show();
        }
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        foreach (Window window in Application.Current.Windows)
        {

```

```
        if (window.GetType() == typeof(MainFrame))
        {
            var wind = new MainWindow();
            wind.frameAuth.Content = new Login();
            wind.Show();
            window.Close();
        }
    }
}
```


Листинг 2 – реализация класса AdminLookRosters

```
public partial class AdminLookRosters : Window
{
    public AdminLookRosters(int id)
    {
        InitializeComponent();
        PlayerVM matchVM = new PlayerVM();
        DataContext = matchVM;
        matchVM.GetTID = id;
        var rr =
matchVM.EHLRecord.PlayersRecords.Where(x=>x.teamID==id);
        table.ItemsSource = rr;
    }
}
```

ПРИЛОЖЕНИЕ М

Листинг 1 - Реализация класса ArticleVM

```
public class ArticleVM
{
    private ICommand _saveCommand;
    private ArticlesRepo _repository;
    private Articles AEntity = null;
    public ArticleRecord ARecord { get; set; }
    public EHLMModel EHLEntities { get; set; }

    public ICommand SaveCommand
    {
        get
        {
            if (_saveCommand == null)
                _saveCommand = new RelayCommand(param =>
SaveData(), null);

            return _saveCommand;
        }
    }
    public ArticleVM()
    {
        AEntity = new Articles();
        _repository = new ArticlesRepo();
        ARecord = new ArticleRecord();
        GetAll();
    }

    public void SaveData()
    {
        if (ARecord != null)
        {
            AEntity.id = Guid.NewGuid();
            AEntity.Image = ARecord.ImagePath;
            AEntity.Header = ARecord.Header;
            AEntity.Creation_Date = (DateTime)DateTime.Now;
            try
            {
                if (AEntity.Image != null && AEntity.Header
!= null)
                {
                    _repository.AddArticle(AEntity);
                    MessageBox.Show("New record
successfully saved.");
                }
                else throw new Exception();
            }
        }
    }
}
```

```

        catch (Exception ex)
        {
            MessageBox.Show("All fields for adding
article should be filled!!! " + ex.InnerException);
        }
        finally
        {
            GetAll();
        }
    }

    public void GetAll()
    {
        ARecord.mangaRecords = new
ObservableCollection<ArticleRecord>();
        _repository.GetAll().ForEach(data =>
ARecord.mangaRecords.Add(new ArticleRecord()
        {
            id = data.id,
            Image = data.Image,
            Header = data.Header,
            Creation_Date = (DateTime)data.Creation_Date
        }))) ;
    }
}

```

Листинг 2 – реализация класса AdminArticles

```
public partial class AdminArticles : Page
{
    public AdminArticles()
    {
        InitializeComponent();
        DataContext = new ArticleVM();
    }

    private void Button_Click_1(object sender, RoutedEventArgs
e)
    {
        foreach (Window window in Application.Current.Windows)
        {
            if (window.GetType() == typeof(MainFrame))
            {
                var wind = new MainWindow();
                wind.frameAuth.Content = new Login();
                wind.Show();
                window.Close();
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Н

Листинг 1 - Реализация класса BonusRecord

```
public class BonusRecord:ViewModels.ViewModelBase
{
    private Guid _id;
    public Guid id
    {
        get
        {
            return _id;
        }
        set
        {
            _id = value;
            OnPropertyChanged("id");
        }
    }
    private string _header;
    public string Header
    {
        get
        {
            return _header;
        }
        set
        {
            _header = value;
            OnPropertyChanged("Header");
        }
    }
    private string _image;
    public string Image
    {
        get
        {
            return _image;
        }
        set
        {
            _image = value;
            OnPropertyChanged("Image");
        }
    }
    #region Image
    private string _imagePath;
    public string ImagePath
    {
        get
        {

```

```

        return _imagePath;
    }
    set
    {
        if (!(value is string))
            return;

        _imagePath = value.ToString();
        OnPropertyChanged("ImagePath");
    }
}
#endregion
#region CommandForPhoto
private readonly RelayCommand _openFileDialog;
public RelayCommand ChoosePhoto
{
    get
    {
        return _openFileDialog ?? (new RelayCommand(obj =>
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Multiselect = false;
            ofd.Filter = "Image
Files|*.jpg;*.jpeg;*.png;*.jfif";
            ofd.ShowDialog();
            ImagePath = ofd.FileName;
        }));
    }
}
#endregion
private ObservableCollection<BonusRecord> _bonusRecords;
public ObservableCollection<BonusRecord> bonusRecords
{
    get
    {
        return _bonusRecords;
    }
    set
    {
        _bonusRecords = value;
        OnPropertyChanged("BonusRecords");
    }
}

private void StudentModels_CollectionChanged(object sender,
NotifyCollectionChangedEventArgs e)
{
    OnPropertyChanged("BonusRecords");
}

private ICommand _getID;

```

```

        public ICommand GetID
        {
            get
            {
                if (_getID == null)
                    _getID = new RelayCommand(param =>
ReturnID((Guid)param), null);

                return _getID;
            }
        }
        public void ReturnID(Guid guid)
        {
            foreach (Window window in Application.Current.Windows)
            {
                if (window.GetType() == typeof(MainFrame))
                {
                    if ((window as MainFrame).Main.Content.GetType()
== typeof(Bonus))
                    {
                        var tt = window as MainFrame;
                        var pageGet = tt.Main.Content as Bonus;
                        var UserId = pageGet.user.id;
                        var telephones = pageGet.ehl.USERS.Select(p
=> new { p.TELEPHONE, p.id});
                        var telephoneUsers =
telephones.Where(p=>p.id == UserId).Single();
                        string telephoneUser =
telephoneUsers.TELEPHONE;
                        try
                        {
                            var result =
pageGet.unitOfWork.OrdersRepo.AddOrders(guid, pageGet.user.id,
"Ordered" + DateTime.Now.ToString("dd/MM/yy"));
                            if (result)
                            {
                                MessageBox.Show("Bonus ordered.
Please wait for our call", "Succeed!", MessageBoxButton.OK);
                                try
                                {
                                    //Just wanted to make SMS -
sending, but if you hadn't Twilio account, it wouldn't work
                                    string AccountSid =
"AC915b8f68024e85aec6e256eecd974b14";
                                    string AuthToken =
"da287dd01531f4c834492edd299b175d";
                                    TwilioClient.Init(AccountSid,
AuthToken);
                                    var message =
MessageResource.Create(

```


Листинг 2 – реализация класса BonusVM

```
private ICommand _saveCommand;
private BonusRepo _repository;
private bonus BEntity = null;
public BonusRecord BRecord { get; set; }
public EHLModel EHLEntities { get; set; }
public ICommand SaveCommand
{
    get
    {
        if (_saveCommand == null)
            _saveCommand = new RelayCommand(param =>
SaveData(), null);

        return _saveCommand;
    }
}
public BonusVM()
{
    BEntity = new bonus();
    _repository = new BonusRepo();
    BRecord = new BonusRecord();
    GetAll();
}

public void SaveData()
{
    if (BRecord != null)
    {
        BEntity.id = Guid.NewGuid();
        BEntity.image = BRecord.ImagePath;
        BEntity.header = BRecord.Header;
        try
        {
            if(BEntity.image!=null &&
String.IsNullOrEmpty(BEntity.header))
            {
                _repository.AddBonus(BEntity);
                MessageBox.Show("New record
successfully saved.");
            }
            else
            {
                throw new Exception();
            }
        }
        catch (Exception ex)
        {

```

```

        MessageBox.Show("Error occurred while
saving. All fields for adding bonus should be filled!" +
ex.InnerException);
    }
    finally
    {
        GetAll();
    }
}

public void GetAll()
{
    BRecord.bonusRecords = new
ObservableCollection<BonusRecord>();
    _repository.GetAll().ForEach(data =>
BRecord.bonusRecords.Add(new BonusRecord()
    {
        id = data.id,
        Image = data.image,
        Header = data.header
    }));
}
}

```

Листинг 3 – реализация класса AdminBonus

```
public partial class AdminBonus : Page
{
    public AdminBonus()
    {
        InitializeComponent();
        DataContext = new BonusVM();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        var window = new AdminAllOrders();
        window.Show();
    }

    private void Button_Click_1(object sender, RoutedEventArgs
e)
    {
        foreach (Window window in Application.Current.Windows)
        {
            if (window.GetType() == typeof(MainFrame))
            {
                var wind = new MainWindow();
                wind.frameAuth.Content = new Login();
                wind.Show();
                window.Close();
            }
        }
    }
}
```

Листинг 4 – реализация класса AdminAllOrders

```
public partial class AdminAllOrders : Window
{
    public AdminAllOrders()
    {
        InitializeComponent();
        FillDataGrid();
    }
    public void FillDataGrid()
    {
        string ConString =
ConfigurationManager.ConnectionStrings["EHLModel"].ConnectionString;
        using (SqlConnection con = new SqlConnection(ConString))
        {
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "select
orders.userid,USERS.NAME,USERS.SURNAME,USERS.TELEPHONE,BONUS.id as
bonusID,BONUS.header from USERS,BONUS,orders where
orders.id=bonus.id and orders.userid=USERS.id";
            cmd.Connection = con;
            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataTable dt = new DataTable("OrdersAll");
            sda.Fill(dt);
            table.ItemsSource = dt.DefaultView;
        }
    }
}
```