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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования

Тема «Программное средство «Diabeta.net» »

Исполнитель

студент 2 курса 6 группы Трошко Валерия Николаевна

(Ф.И.О.)

Руководитель работы Радиванович Д. А.

(подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

(подпись)

Минск 2021

# Содержание

|  |  |
| --- | --- |
| Введение | 4 |
| 1. Обзор литературы и аналогов | 5 |
| * 1. Everyone Piano | 5 |
| * 1. Dream Computer Piano 2.0 | 5 |
| * 1. PC 73 Virtual Piano Keyboard | 6 |
| * 1. Аналогичность и различие программных средств | 7 |
| 1. Анализ требований к программному средству и разработка функциональных требований | 8 |
| * 1. Режим свободной игры | 8 |
| * 1. Режим обучения | 9 |
| 1. Проектирование и реализация программного средства | 10 |
| * 1. Архитектура проектируемого программного средства | 10 |
| * 1. Разработка базы данных | 10 |
| * 1. Общая структура проекта | 12 |
| * 1. Структура классов проекта | 13 |
| 1. Описание интерфейсов программного средства и его тестирование | 15 |
| * 1. Описание интерфейсов программного средства | 15 |
| * 1. Тестирование программного средства | 20 |
| 1. Руководство по установке и использованию | 22 |
| * 1. Установка программного средства | 22 |
| * 1. Руководство пользователя по работе с программным средством | 23 |
| Заключение | 28 |
| Список использованной литературы | 29 |
| Приложение 1. Листинг скриптов | 30 |

# Введение

В современном мире информационные технологии развиваются с невероятной скоростью. Так, за относительно небольшой промежуток времени истории человечества они стали играть немаловажную роль во всех сферах жизни общества. С развитием технологий человеку становятся доступны все новые и новые возможности, о которых ранее приходилось только мечтать.

Стоит отметить, среди трендов последних лет очень четко выделяется направление самоорганизации. Приложения для повышения продуктивности занимают все более высокие позиции в рейтингах по количеству скачиваний в PlayMarket и AppStore. Тренд очевиден: современный человек сегодня – это человек, стремящийся контролировать и анализировать свою жизнь. И, безусловно, такой человек не откажется от удобного приложения, которое позволит контролировать дневной рацион питания и норму калорий. В связи с этим была выбрана главная цель курсового проекта: разработка программного средства, которое сочетало бы в себе возможности современных компьютеров и ноутбуков для организации обучения пользователя игре на фортепиано в развлекательной форме. Определяющим фактором выбора инструмента для разработки программного средства стала огромная любовь автора данного курсового проекта к игре на фортепиано, что и подтолкнуло его к созданию простого и современного приложения.

Актуальность данной темы заключается в том, что многие люди заинтересованы в том, чтобы научиться играть несложные мелодии на фортепиано, однако не у всех есть возможность приобретения дорогостоящих музыкальных инструментов или посещения преподавателей. Исходя из этого при разработке программного средства были поставлены следующие задачи:

* разработка десктопного приложения, позволяющее использовать компьютерную клавиатуру в качестве клавиатуры фортепиано;
* разработка удобного и приятного пользовательского интерфейса;
* создание уроков, которые дадут пользователю начальные навыки игры на музыкальном инструменте;
* разработка функций, позволяющих отслеживать прогресс пользователя, а также воспроизводить и сохранять музыкальные фрагменты, сыгранные пользователем.

Программное средство рассчитано в первую очередь на тех, у кого есть большое желание научиться играть на фортепиано, однако нет каких-либо знаний в этой области. Но также приложение подойдёт и для профессионалов, которые смогут обучиться новым произведениям или поделиться своим творчеством с другими пользователями.

# Обзор литературы и аналогов

При изучении программ-прототипов было выявлено несколько схожих по функциональности программ. Все программные средства представлены в качестве приложений для поддержания здоровья. Результат анализа представлен ниже.

## Glucose Buddy Diabetes Tracker

Данное приложение было разработано под мобильные операционные системы IOS и Android. Имеет следующие возможности:

* + Легко регистрировать уровень глюкозы крови, лекарства, приемы пищи, в одной записи
  + Отслеживание изменения уровня сахара в крови, инсулина, веса, артериального давления, A1C и многого другого!
  + Добавление заметок к вашим записям для дальнейшего использования
  + Запись своих блюд, используя нашу обширную базу данных продуктов питания

Интерфейс приложения представлен на рисунке 1.1.

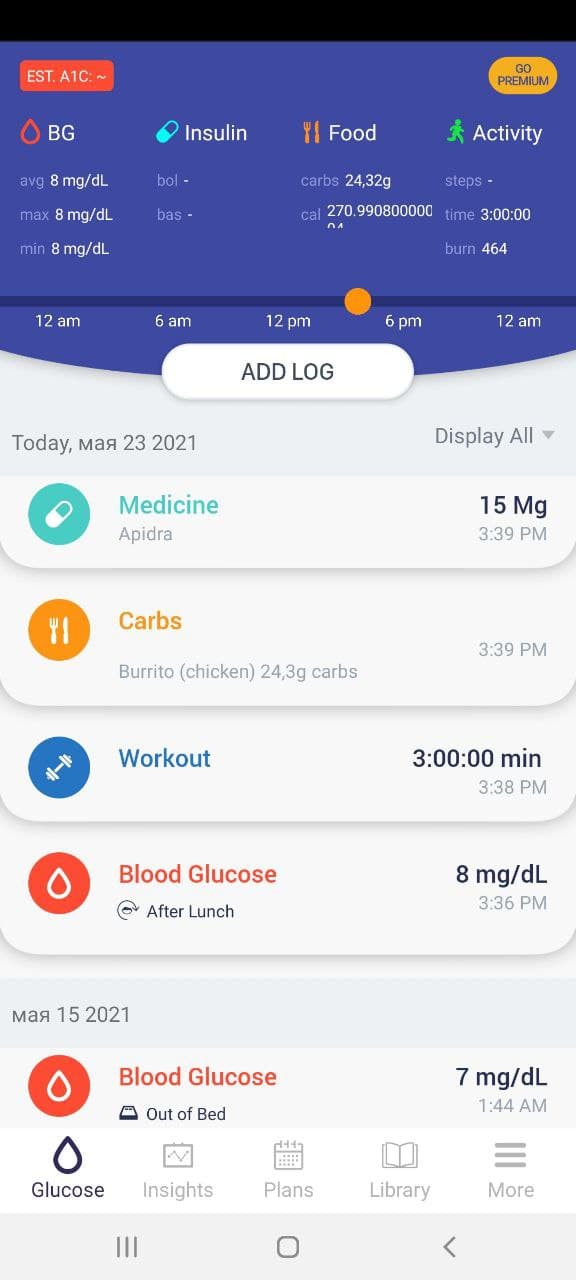


Рисунок 1.1 − Интерфейс приложения Glucose Buddy Diabetes Tracker

Из недостатков необходимо отметить не самый приятный и интуитивно непонятный дизайн программного средства. Большинство возможностей доступны только в платной версии приложения. Отсутствие контроля водного баланса.

## Диабет

Данное приложение так же было разработано под мобильные операционные системы IOS и Android. Имеет следующие возможности:

* Ведение дневника контроля глюкозы
* Ведение дневника питания
* Точные расчеты ХЕ на еду (самая большая база продуктов + возможность добавлять свои продукты)
* Расчет доз инсулина (в том числе дробный инсулин для помпы)
* Экспорт дневника в .pdf и .xls, печать дневника, отправка прямо из приложения в любой меседжер или на почту.

Внешний вид программы представлен на рисунке 1.2.

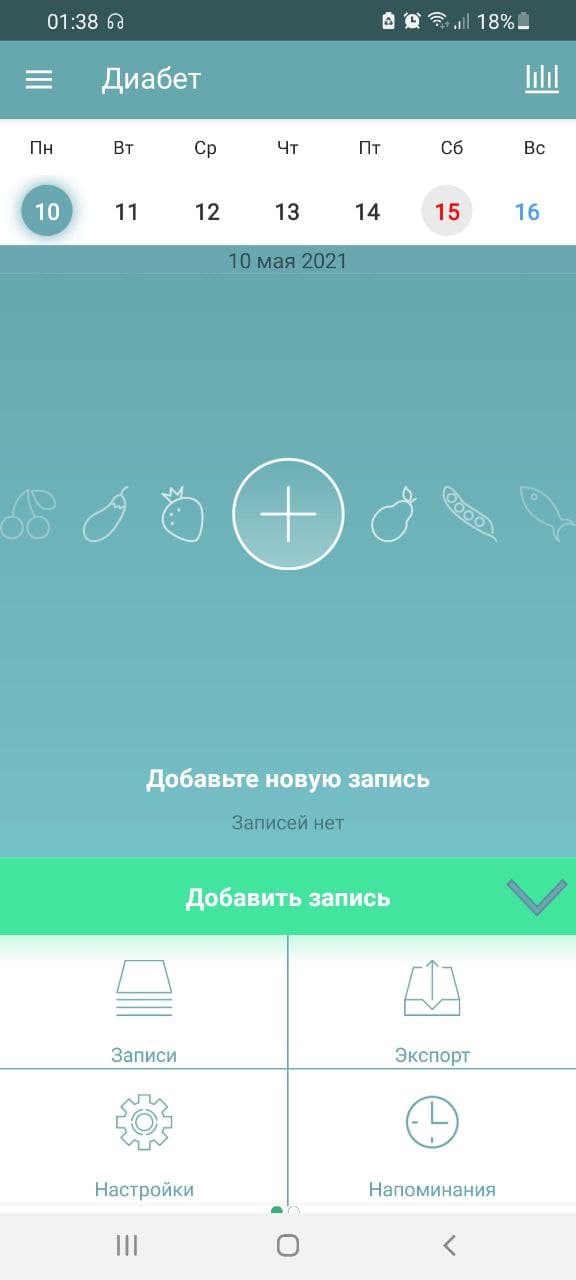


Рисунок 1.2 — Интерфейс приложения Диабет

Диабет, как и приложение, рассмотренное ранее, не отличается удобством в использовании. Его интерфейс выглядит устаревшим и однозначно будет не очень приятен пользователю. В нём так же отсутствует функция записи употреблённых лекарств. Отсутствие контроля водного баланса.

## Diabetes Tracker

Дневник контроля глюкозы крови и питания – важная вещь в жизни диабетика. Приложение Diabetes Tracker создано в помощь как тем, кто впервые столкнулся с этим диагнозом, так и давно болеющим людям. Функционал:

* Удобный ежедневник с графами для внесения результатов многократных измерений
* Заметки, куда можно вписывать любую другую важную информацию: объем текста – 140 знаков
* Архив, который поможет при необходимости получить давние сведения, сохраненные ранее трех месяцев назад
* Статистика – раздел, где анализируются и интерпретируются результаты контроля сахара
* Цветные графики - "сахарная кривая" визуального контроля измерений сахара крови за разные промежутки времени: 2 недели, месяц, 3 месяца. Пользователям также доступен график изменений HbA1c за год
* Интеграция с Google Drive - способ быстро и надёжно разместить в облачном хранилище все результаты измерений сахара крови
* Настройки интерфейса

Главное окно программы представлено на рисунке 1.3.



Рисунок 1.3 — Интерфейс приложения Diabetes Tracker

К недостаткам можно отнести отсутствие контроля водного баланса и то, что приложение поддерживает только один язык.

## Аналогичность и различие программных средств

Схожесть рассмотренных программных средств заключается в том, что все они позволяют следить за нормой КБЖУ, употреблением медицинских препаратов и инсулина.

Главные отличия программ кроются в их функциональности. Так, например, в приложениях Diabetes Tracker и Glucose Buddy Diabetes Tracker доступны заметки, в приложении Diabetes Tracker пользователь может сам регулировать внешний вид приложения, а в приложении Диабет существует возможность сохранения в .pdf и .xls.

Таким образом, можно сделать вывод о том, что в каждом приложении есть свои плюсы и минусы, что и было учтено при выполнении данного курсового проектирования.

# Анализ требований к программному средству и разработка функциональных требований

Для того чтобы приступить к этапу разработки проекта, необходимо четко сформулировать функциональные требования к программному средству.

## 2.1 Определение требований к программному средству

Функционально ПС должно выполнять следующие задачи:

* регистрация и авторизация пользователей;
* сохранение информации в базе данных;
* для администратора одобрение или отклонение предложенных продуктов;
* для пользователя изменение личных данных;
* для пользователя учёт выпитых медикаментов,
* для пользователя водного и пищевого баланса;
* уведомление пользователя об высоком или низком уровне сахара;
* для администратора добавление новых продуктов и рецептов.

## 2.2 Описание средств разработки

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2019;
* программная платформа .NET Framework 5.0.202;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология ADO.NET;
* Microsoft SQL Server.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс – Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом. В основе графической технологии WPF лежит мощная инфраструктура, основанная на DirectX. Это является одним из основных отличий WPF от более ранней технологии создания пользовательских интерфейсов – Windows Forms.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом – С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Если различные механизмы работы с БД в С#. Т.к. моя база данных небольшая, мой выбор пал на ADO.Net. ADO.Net предоставляет самый прямой способ доступа к данным в .Net Framework.

## 2.3 Описание функциональности программного средства

Описание функциональности программного средства представлено с помощью UML-диаграммы вариантов использования (Приложение А).

Спецификация функциональности программного средства представлена в следующем пункте

## 2.4 Спецификация функциональных требований

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Подробно о базе данных описано в главе 3.

В программном средстве «Diabeta.net» при запуске необходимо реализовать регистрацию и авторизацию пользователей для дальнейшего использования приложения. Для авторизации входными параметрами являются логин и пароль пользователя, которые содержатся в базе данных. Для того чтобы зарегистрироваться необходимо ввести логин, пароль, фамилию, имя, возраст, рост, вес, пол, желаемый уровень сахара, цель и уровень подвижности. Введенные данные, успешно прошедшие валидацию, заносятся в базу данных, а пользователь получает доступ к приложению.

При авторизации администратора он может на странице «*Одобрить*» просматривать продукты, добавлять и удалять, при нажатии на кнопку «*Продукты*» может просматривать все продукты и добавлять новые. На странице «*Рецепты»* просматривать имеющиеся рецепты, при нажатии на «*Добавить рецепт*» администратор может создать и добавить новый рецепт.

При авторизации пользователя он может на главной странице просматривать и отмечать все съеденные продукты и блюда, весь инсулин и все таблетки, которые употреблял, уровень сахара и выпитые стаканы воды. На странице «*Инфо»* он может просмотреть личную информацию и редактировать её. На странице «*Рецепты*» он может увидеть существующие рецепты и просмотреть их. На страницах статистик пользователь может просмотреть статистику по весу и статистику по уровню сахара в крови.

При авторизации администратора он может на странице «*Товары*» просматривать все товары от всех продавцов и удалять их. На странице «*Заказы»* просматривать заказы и удалять их. На странице «*Продавцы*» просматривать всех продавцов и удалять их. На странице «*Клиенты*» просматривать всех клиентов и удалять их.

Кнопка для выхода из аккаунта возвращает пользователя к окну авторизации, а кнопка для выхода из приложения закрывает приложение.

Все данные о продуктах, рецептах, таблетках и сахаре в крови должны заноситься в базу данных в соответствующие таблицы.

# Проектирование программного средства

## Архитектура проектируемого программного средства

При создании программного средства использовался архитектурный паттерн Model-View-ViewModel (MVVM). Он был представлен архитектором программного обеспечения компании Microsoft Джоном Госсманом в 2005 году. Одной из главных целей шаблона является отделение и абстрагирование видимого пользовательского интерфейса от логики представления. MVVM состоит из трех компонентов: модель (Model), представление (View) и модель представления (ViewModel). Наглядное представление паттерна представлено на рисунке 3.1.

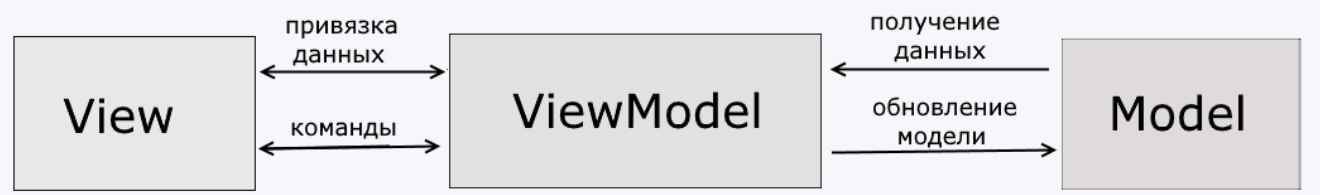


Рисунок 3.1 − Наглядное представление MVVM

Модель (Model) — это субъект, который описывает используемые в приложении данные. Это может быть что угодно, будь то обычный продукт или же сложный рецепт. Представление (View) — это графический элемент управления или набор элементов управления, отвечающих за отображение данных на экране. Модель представления (ViewModel) — это и есть та магия, стоящая за всем. Она содержит логику пользовательского интерфейса, команды, события и ссылку на модель. В MVVM модель преставления не обязана обновлять непосредственно данные представления. Благодаря мощному механизму привязки, представленному в WPF, пользовательский интерфейс обновляется, как только обновляется модель преставления. Для этого модель преставления должна реализовывать интерфейс INotifyPropertyChanged и запустить событие PropertyChanged [4].

## Разработка базы данных

Для обеспечения работоспособности программного средства была разработана база данных, что являлось обязательным условием курсового проекта. База данных была тщательно сконструирована до начала написания самого приложения. Для ее создания использовалась реляционная система управления базами данных (РСУБД) MS SQL.

Таким образом была разработана база данных KP\_DataBase, которая включает в себя таблицы Users, Daily\_Food, Recipe, Prod\_Rec, Products, Type\_of\_Food, Daily\_Insulin, Type\_of\_Insulin, Daily\_Pill, Daily\_Cal, Daily\_Water, History, History\_Blood\_Sugar, Products\_Awaiting\_Approval. Схема базы данных представлена на рисунке 3.2

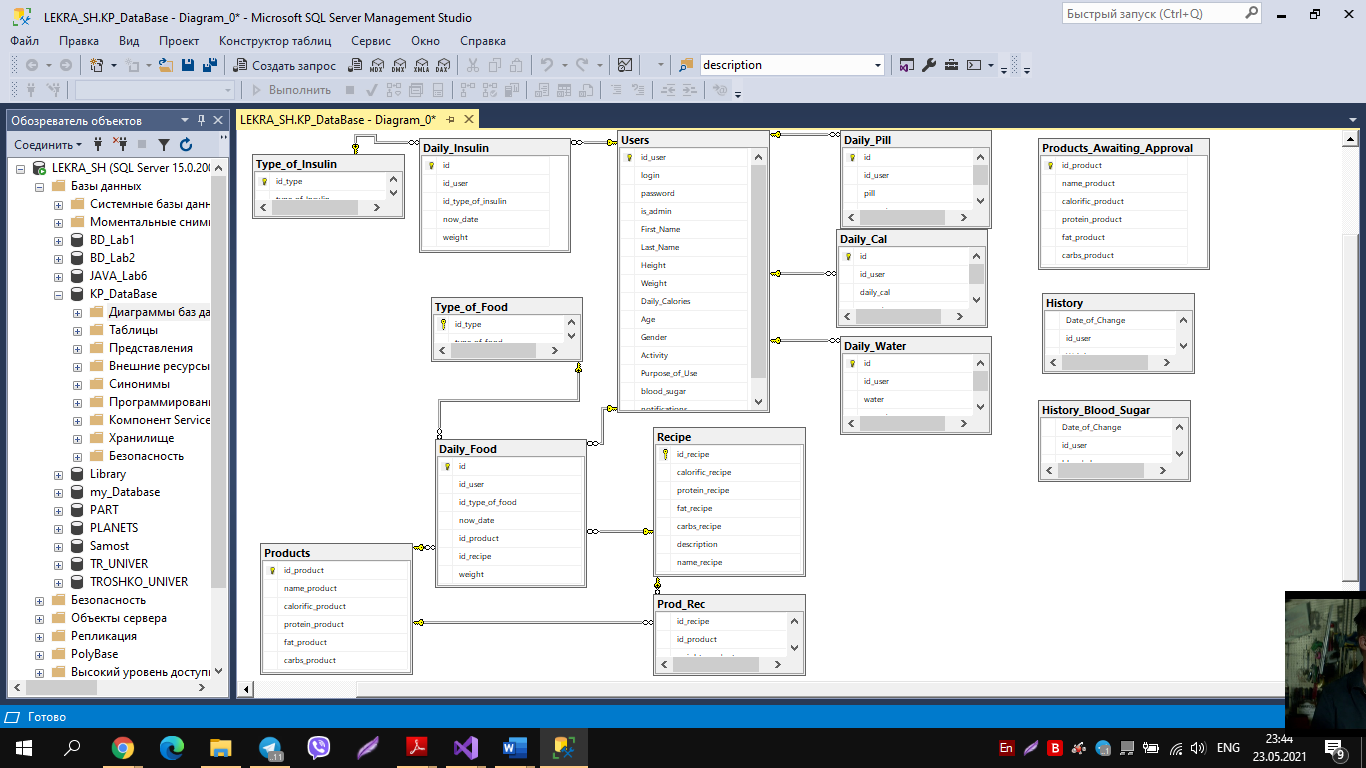


Рисунок 3.2 — Схема базы данных KP\_DataBase

Для хранения информации о пользователях была создана таблица Users. Включает 15 столбцов: id\_user – id пользователя, login – логин пользователя, password – пароль пользователя, при записи в базу данных хэшируется, isadmin – 0 – пользователь, 1 - администратор, First\_Name – имя пользователя, Last\_Name – фамилия пользователя, Height – рост, Weight – вес, Daily\_Calories – дневная норма калорий, Age – возраст, Gender – пол, Activity – коэффициент активности, Purpose\_of\_Use – цель использования, blood\_sugar – коэффициент уровня сахара в крови, notifications – поле доступа для уведомлений.

Таблица Daily\_Food используется для хранения съеденной пищи пользователем в течении дня.

Таблица Recipe используется для хранения информации о рецептах. Включает 7 столбцов: id\_recipe – id рецепта, name\_ recipe – название рецепта, calorific\_ recipe – калорийность рецепта на 100г, protein\_ recipe - белки рецепта на 100г, fat\_ recipe – жиры рецепта на 100г, carbs\_ recipe - углеводы рецепта на 100г, description – способ приготовления.

Таблица Prod\_Rec используется для связи между таблицами Recipe и Products. Используется для определения того, какие ингредиенты входят в состав блюда. Включает 3 столбца: id\_recipe – id рецепта, id\_product – id продукта, weight\_product – вес продукта, необходимого для рецепта.

Таблица Products используется для хранения продуктов. Включает 6 столбцов: id\_product – id продукта, name\_product – название продукта, calorific\_product – калорийность продукта на 100г, protein\_ product – белки продукта на 100г, fat\_ product – жиры продукта на 100г, carbs\_ product - углеводы продукта на 100г.

Таблица Type\_of\_Food используется для хранения данных о типах приема пищи. Включает 2 столбца: id\_type – id приема пищи, type\_of\_food – тип приема пищи (завтрак, обед, ужин, перекус).

Таблица Daily\_Cal хранит оставшееся число калорий пользователя на день.

Таблица Daily\_Water используется для хранения выпитой воды пользователем в течении дня.

Таблица Daily\_Pill используется для хранения выпитых медикаментов пользователем в течении дня.

Таблица Daily\_Insulin используется для хранения сделанных уколов пользователем в течении дня.

Таблица Type\_of\_Insulin используется для хранения данных о типах уколов. Включает 2 столбца: id\_type – id приема инсулина, type\_of\_insulin – тип укола инсулина (ночной, дневной).

Таблица Product\_Awaiting\_Approval используется для хранения добавленных пользователем новых продуктов. Элементы жду подтверждения администратора.

Таблица History используется для хранения истории веса пользователя. Включает 3 столбца: Date\_of\_Change – дата изменения веса,id\_user- id пользователя, Weight – обновленный вес. Таблица History автозаполняется, когда срабатывает триггер таблицы Users.

Таблица History\_Blood\_Sugar используется для хранения истории уровня сахара пользователя. Включает 3 столбца: Date\_of\_Change – дата изменения веса,id\_user- id пользователя, blood\_shugar – обновленный Уровень сахара. Таблица History автозаполняется, когда срабатывает триггер таблицы Users.

В базе данных между таблицами Users и Daily\_Insulin реализована связь один-ко-многим, т. к. одному пользователю может соответствовать несколько записанных уколов, а каждый укол в таблице Daily\_Insulin соответствует только одному пользователю. Аналогичная связь реализована в таблицах Users и Daily\_Pill и ещё много где. Схематическое представление связи представлено на рисунке 3.3.

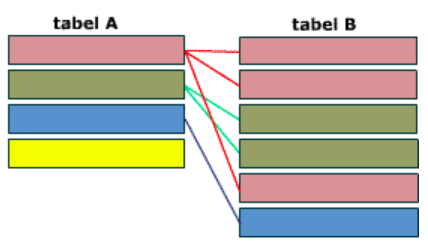


Рисунок 3.3 − Структура связи один-ко-многим

## Обобщенная структура проекта

В общей сложности проект включает в себя 4 папки. Его структура представлена на рисунке 3.4.

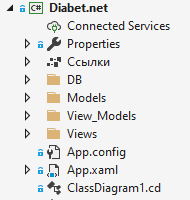


Рисунок 3.4 − Структура проекта

В папке Models хранятся классы, описывающие логическую обработку данных проекта.

Папка View\_Models содержит в себе классы со свойствами и командами, которые связаны механизмом привязки с объектами папки Views.

В папке Views содержатся элементы пользовательского интерфейса, которые пользователь сможет увидеть на экране.

Содержимое папки DB представляет из себя все взаимодействия с базами данных. Приложение включает в себя 13 окон и 12 страниц. Связи между ними представлены на рисунке 3.5.

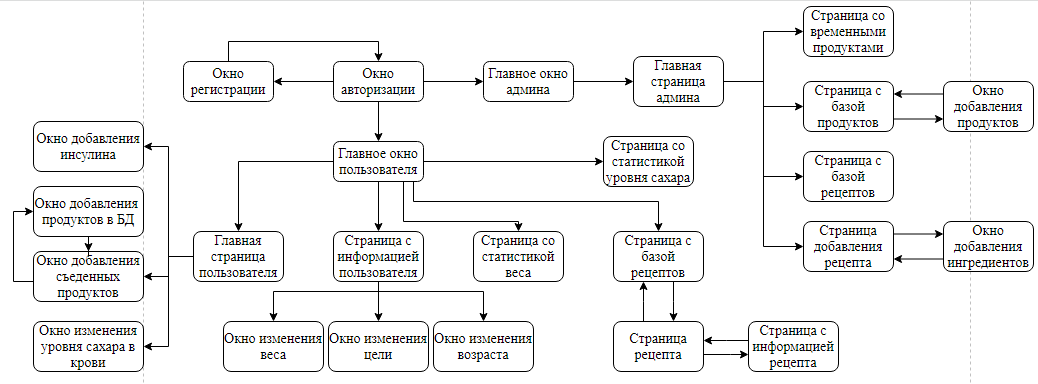
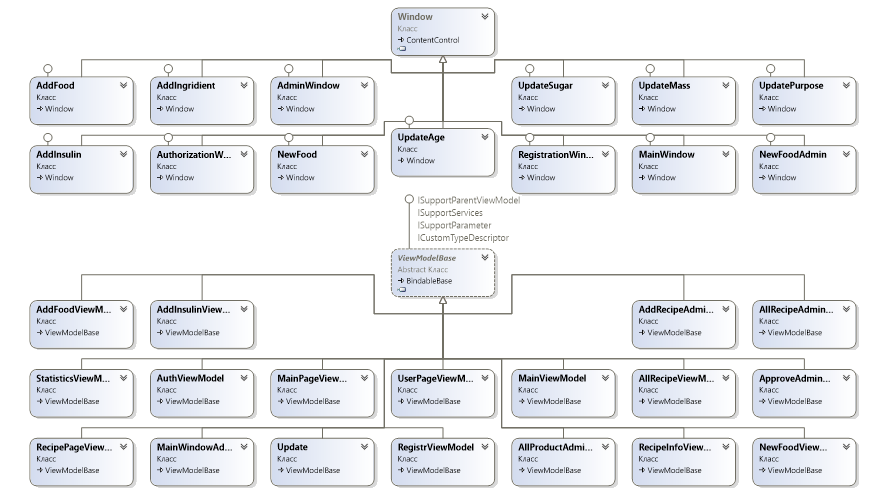
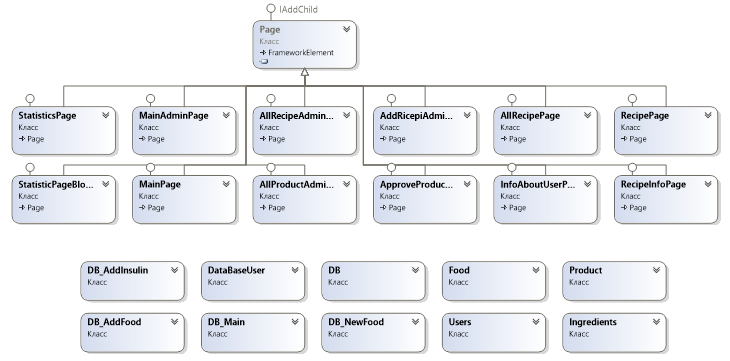


Рис. 3.5 – Схема связи между окнами

## Структура классов проекта

Диаграмма классов представлена на рисунке 3.6.





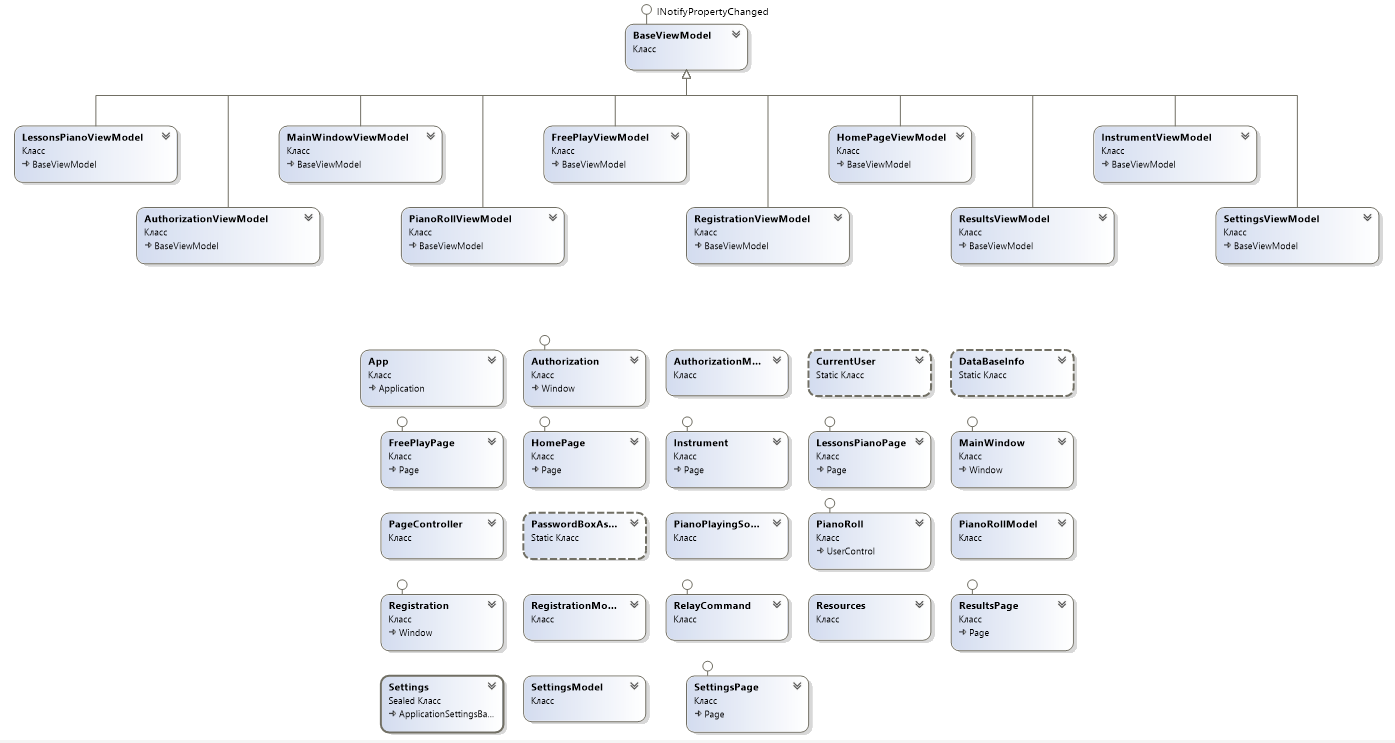


Рисунок 3.5 − Диаграмма классов программного средства

Класс BaseViewModel является родительским для всех классов модели представления программы, в котором реализован интерфейс INotifyPropertyChanged, оповещающий механизм привязки о том, что свойство представления или модели представления изменилось. Это сделано для того, чтобы каждый раз не дублировать реализацию интерфейса в классах модели представления.

Класс RelayComman предназначен для возможности использования команд в приложении.

Классы AuthorizationViewModel и AuthorizationModel отвечают за обработку введенных пользователем данных и их логическую валидацию соответственно в окне авторизации.

Классы RegistrationViewModel и RegistrationModel отвечают за обработку введенных пользователем данных и их логическую валидацию соответственно в окне регистрации.

Класс PageController реализует переходы страниц в окне MainWindow.

Класс DataBaseInfo реализует методы получения данных из базы данных.

Класс CurrentUser хранит информацию о текущем пользователе.

Класс MainWindowViewModel предназначен для реализации навигации пользователя по окнам приложения.

Класс HomePageViewModel содержит в себе команду, реализующую переход на страницу InstrumentPage.

Класс InstrumentViewModel реализует две команды, позволяющие переходить к двум основным страницам приложения: FreePlayPage и LessonsPinaoPage.

Классы PianoRollViewModel и PianoRollModel созданы специально для пользовательского элемента управления PianoRoll, в которых определена логика взаимодействия пользователя с виртуальной фортепианной клавиатурой.

Класс FreePlayViewModel является одним из основных программных модуля приложения. Вместе с классом PianoPlayingModel они реализуют логику режима «Свободная игра».

Класс LessonsPianoViewModel также является одним из главных классов программы. Совместно с классом PianoPlayingModel они реализуют логику работы режима «Обучение».

Класс ResultsViewModel предназначен для отображения результатов пользователя на странице ResultsPage.

Классы SettingsViewModel и SettingsModel содержат в себе описание логики функций «Сброс прогресса» и «Удаление профиля».

Помимо БД и описания классов, покажите какие-нибудь функции. Код оформляете в рамку и подписываете как рисунок. Все подряд не нужно, только те моменты, которые на ваш взгляд покажут «уровень» вашего функционала, т. е. наиболее сложные.

# Описание интерфейсов программного средства, его тестирование и анализ полученных данных

## Описание интерфейсов программного средства

Сразу после запуска приложения перед пользователем открывается окно авторизации, в которое требуется ввести логин и пароль. Внешний вид окна представлен на рисунке 4.1.

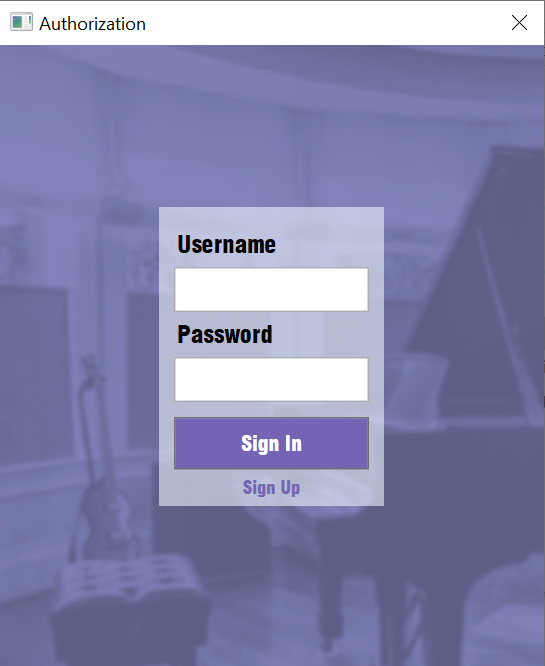


Рисунок 4.1 − Внешний вид окна авторизации

Из окна авторизации пользователи, которых нет в базе данных приложения, могут зарегистрироваться, нажав на кнопку «Sign Up». После того, как это произойдет, откроется окно регистрации новых пользователей, интерфейс которого представлен на рисунке 4.2.

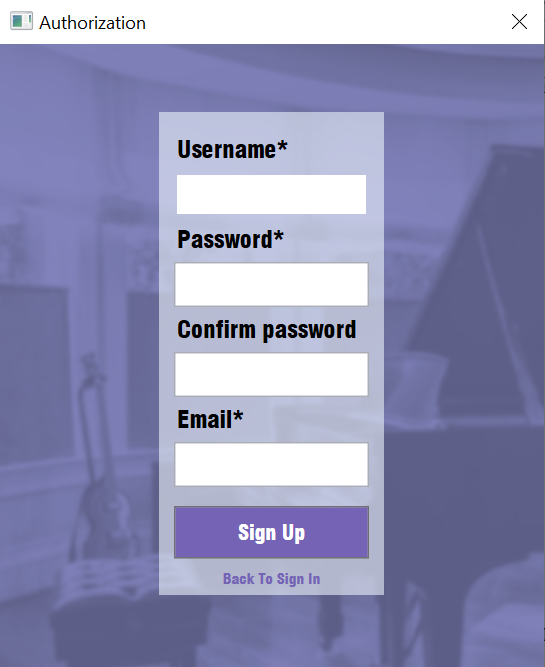


Рисунок 4.2 − Интерфейс окна регистрации

Также в окне регистрации есть функция возврата к окну авторизации, которую реализует элемент управления с надписью «Back To Sign In».

После успешной авторизации открывается главное окно приложения, которое встречает пользователя приветствием с его именем. Внешний вид главного окна представлен на рисунке 4.3.

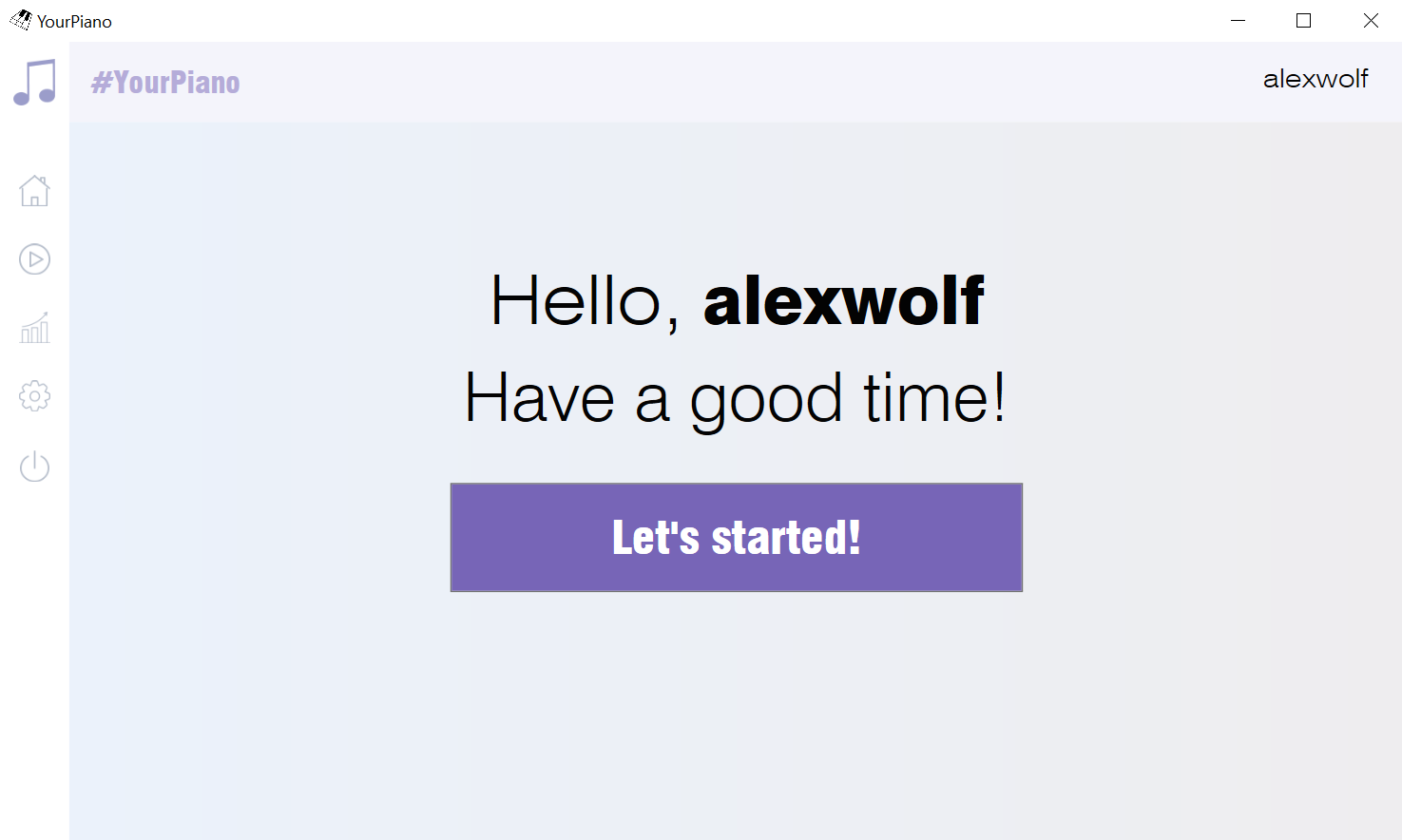


Рисунок 4.3 − Внешний вид главного окна приложения

Также стоит отметить, что вышеупомянутое окно по сути является единственным окном приложения, т. к. при открытии других интерфейсов сменяться будут не окна, а лишь содержимое контейнера Frame главного окна.

При нажатии кнопки «Let’s started!» открывается страница, на которой пользователь может выбрать один из режимов «Свободная игра» или «Обучение». На рисунке 4.4 представлен внешний вид этой страницы.

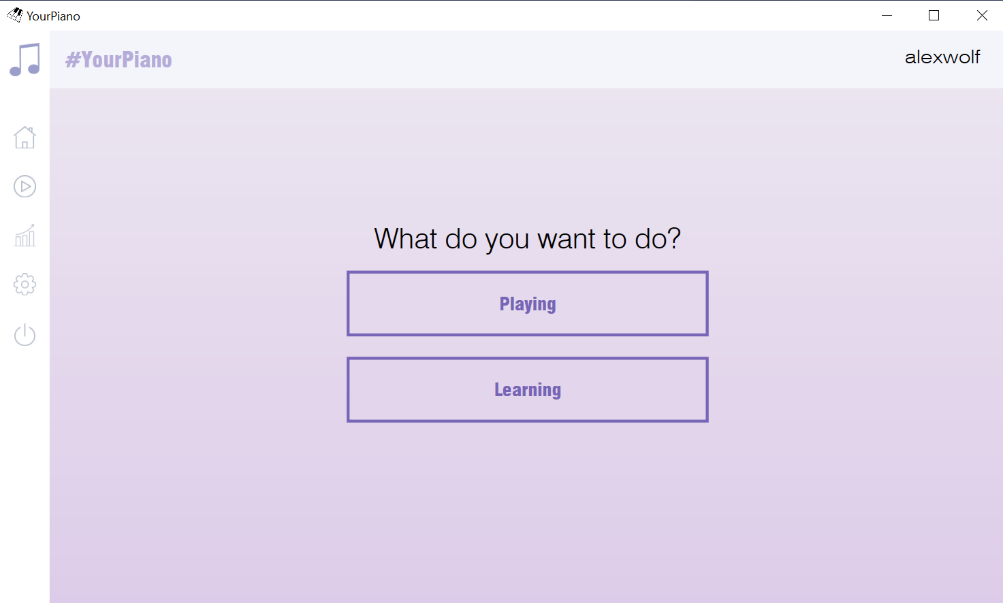


Рисунок 4.4 − Окно выбора режима игры

При выборе режима «Свободная игра» перед пользователем появится соответствующее окно, изображенное на рисунке 4.5. На нем изначально пользователь сможет увидеть лишь клавиатуру фортепиано, а также панель инструментов в верхней части страницы.

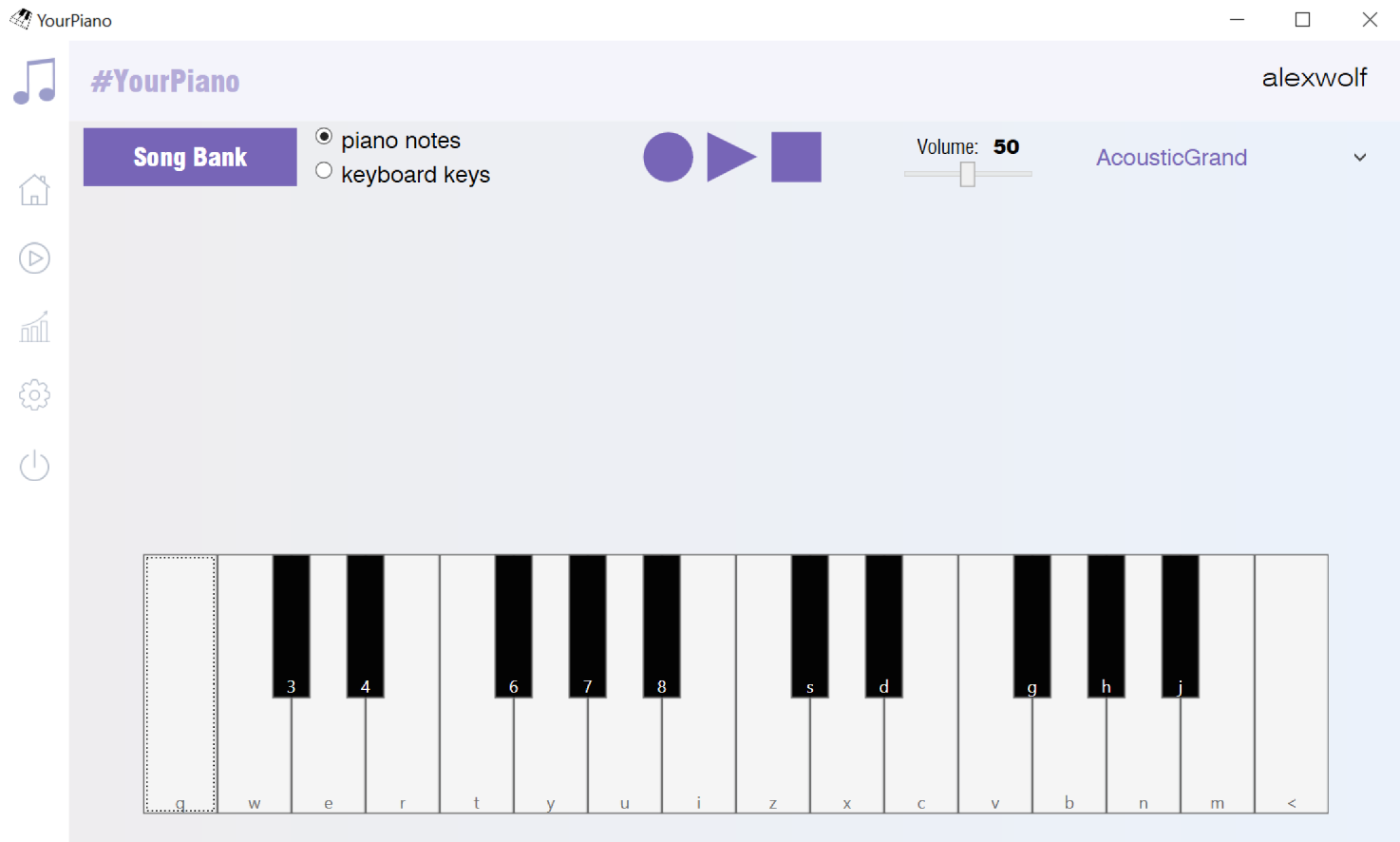


Рисунок 4.5 − Окно режима «Свободная игра»

При нажатии на кнопку «Song Bank» пользователь сможет выбрать желаемое произведение. После выбора произведения интерфейс страницы дополнится панелью, на которой можно увидеть посередине текущую ноту, а справа от нее следующую. При дальнейшем воспроизведении слева от текущей ноты будет появляться и предыдущая. Также имеется элемент управления ProgressBar, показывающий ход воспроизведения композиции. По краям страницы имеются элементы управления, позволяющие управлять воспроизведением вручную. Рисунок 4.6 отображает внешний вид режима с выбранной композицией.

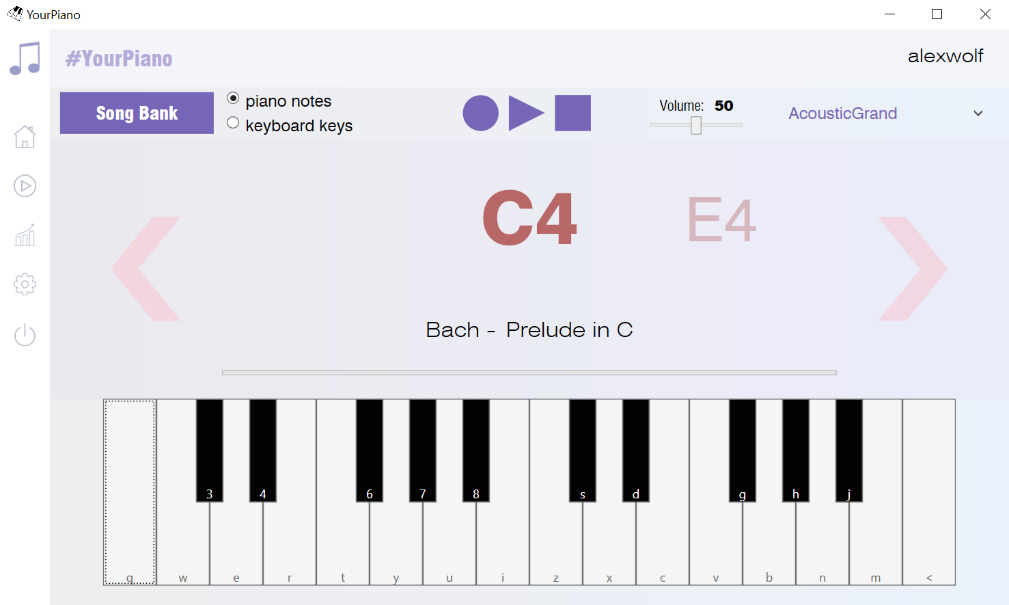


Рисунок 4.6 − Окно режима «Свободная игра» с выбранной композицией

При переходе во второй основной режим «Обучение» пользователя встречает список из уроков, которые он может выбрать для прохождения. Этот список представлен на рисунке 4.7.

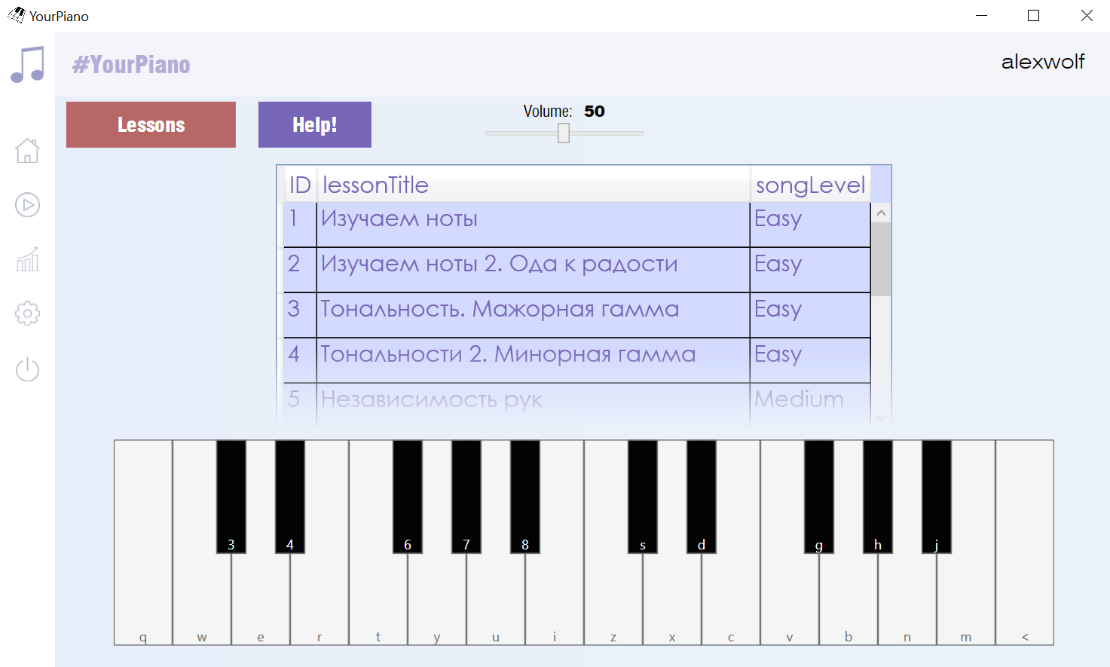


Рисунок 4.7 − Окно режима «Обучение»

При выборе необходимого урока перед пользователем откроется текстовый блок с некоторыми теоретическими сведениями, изображенный на рисунке 4.8.

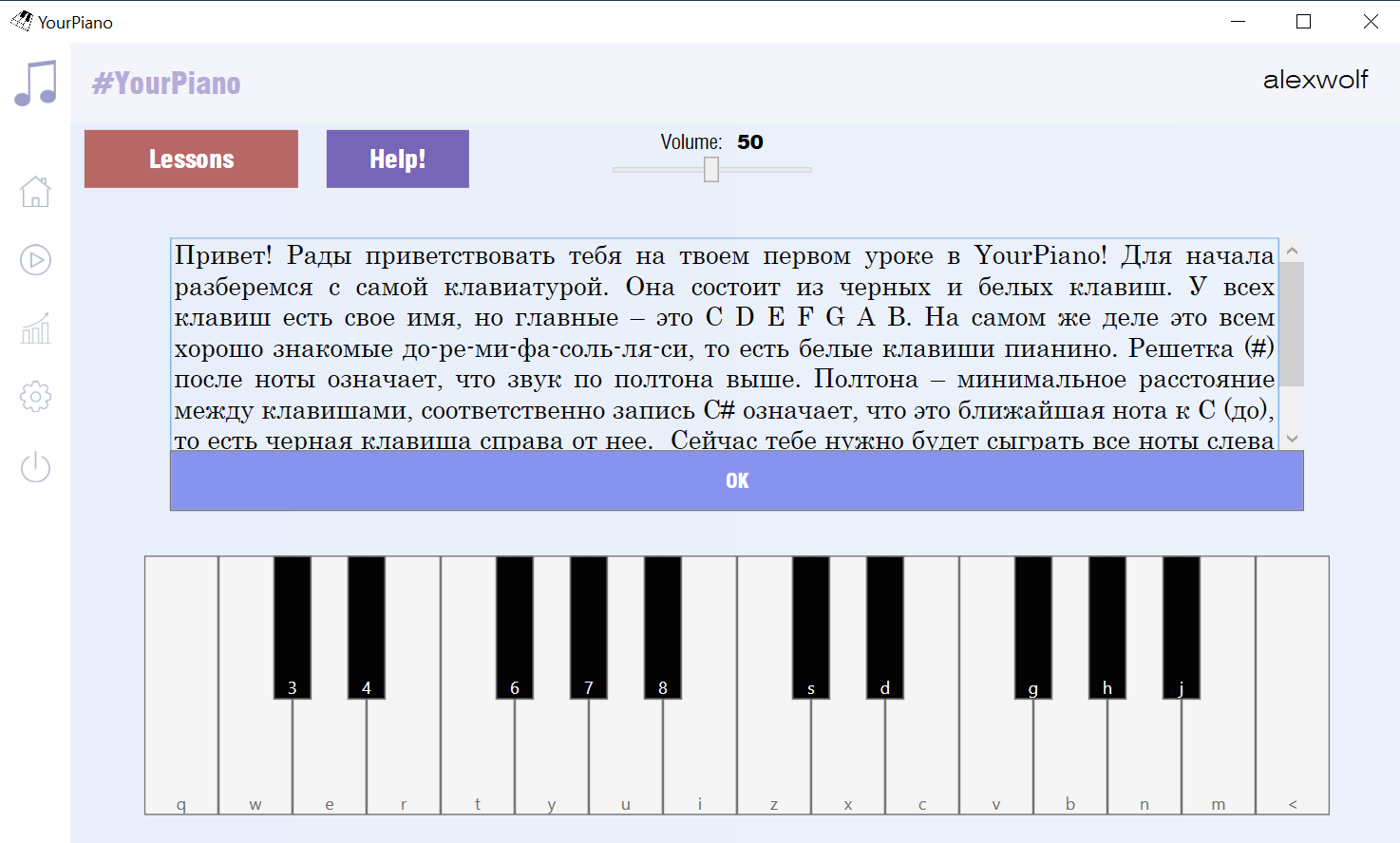


Рисунок 4.8 − Окно режима «Обучение» с текстовым блоком урока

После прочтения теоретических сведений пользователь сможет начать прохождение урока, т. к. перед ним откроется панель с нотами, которая уже встречалась ранее в режиме «Свободной игры». Внешний вид страницы представлен на рисунке 4.9.

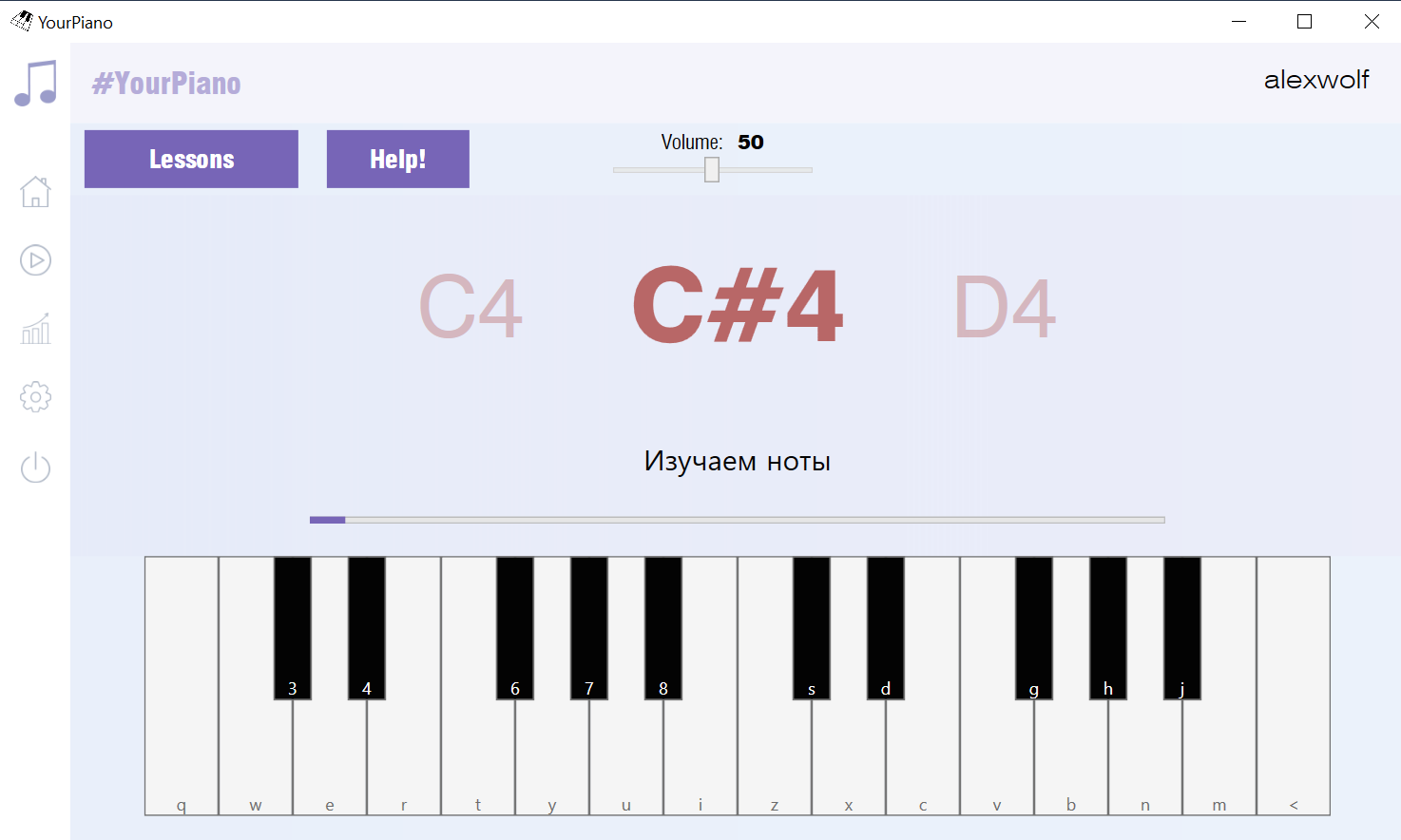


Рисунок 4.9 − Окно режима «Обучение» в процессе прохождения урока

Также в приложении присутствует страница «Прогресс», на которой статистика пользователя. Интерфейс страницы представлен на рисунке 4.10.



Рисунок 4.10 − Окно страницы «Прогресс»

Последней страницей приложения является страница «Настройки», в которой пользователь может сбросить текущий прогресс или полностью удалить свой профиль. Интерфейс окно изображен на рисунке 4.11.

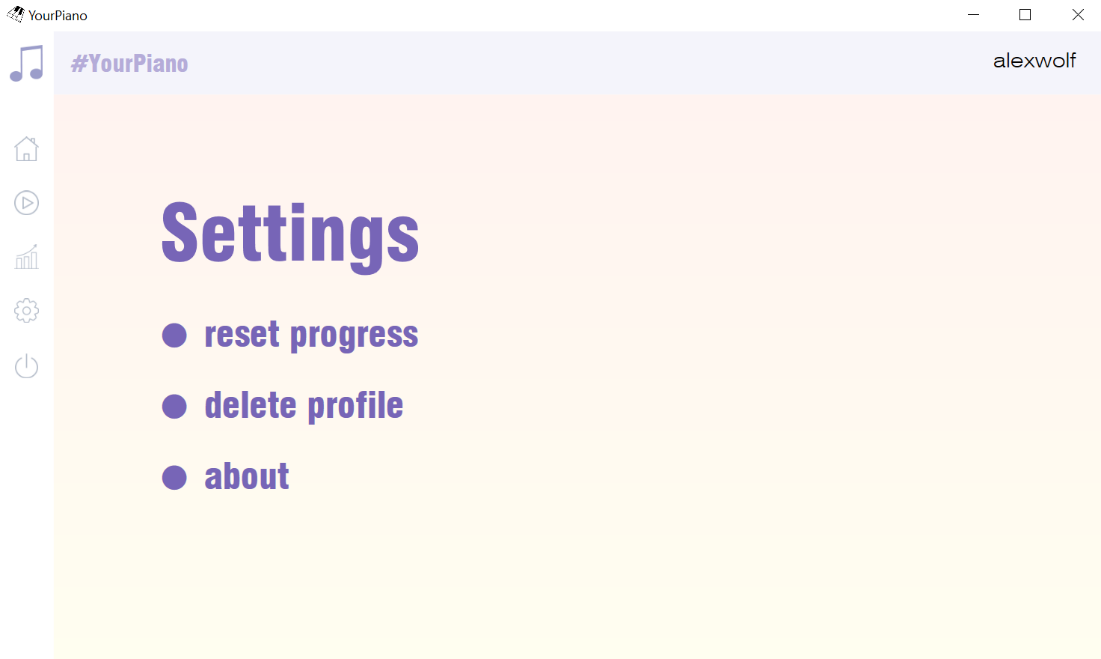


Рисунок 4.11 − Окно страницы «Настройки»

## Тестирование программного средства и анализ полученных данных

Для проверки работоспособности программного средства было проведено его тщательное тестирование, в ходе которого осуществлялась проверка заполняемости базы данных, корректность извлекаемых из базы данных значений, а также реакция программы на вводимые, заведомо неверные, данные. Было проработано множество ситуаций, которые могут возникнуть в ходе работы с программным средством.

Так, при входе осуществляется проверка на действительные логин и пароль. При вводе неверных данных или оставленных незаполненных полей соответствующие поля становятся красными, как показано на рисунке 4.12.

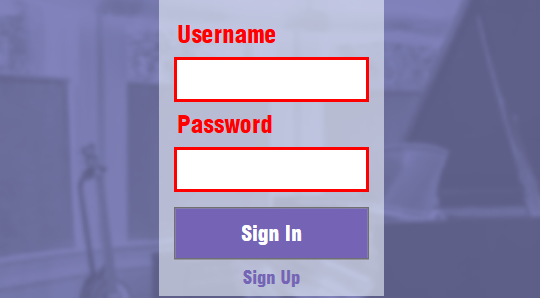


Рисунок 4.12 − Неверно введенные данные в окне авторизации

При рассмотрении исключительных ситуаций в окне регистрации также была организована проверка вводимых пользователем данных. Осуществляется она при помощи регулярных выражений. Таким образом, если хотя бы одно из полей ввода будет заполнено некорректно, то в режиме реального времени оно будет подсвечиваться красным цветов и зеленым цветом в противном случае. Если же будет нажата кнопка регистрации при некорректно введенных данных пользователь получит сообщение об ошибке. Наглядное представление ситуации показано на рисунке 4.13.

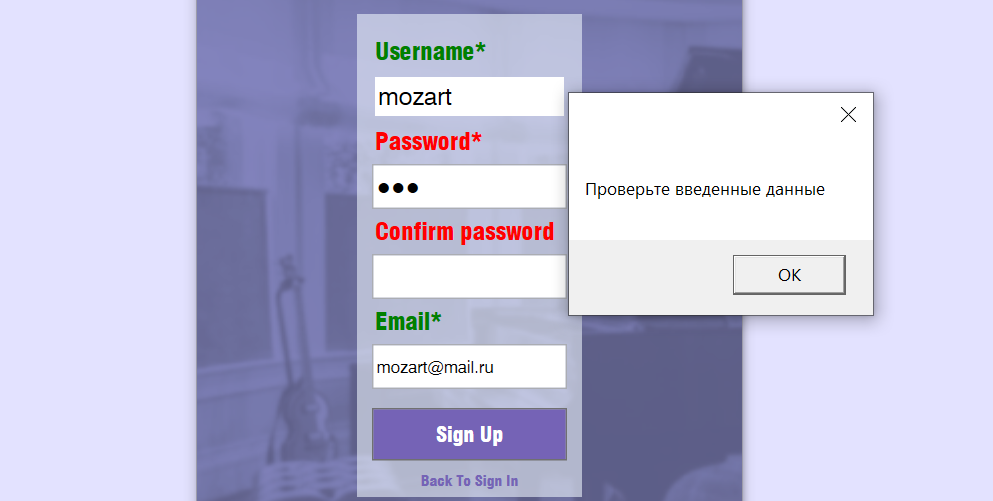


Рисунок 4.13 − Неверно введенные данные в окне регистрации

Еще одна ситуация была рассмотрена в режиме «Свободной игры». Пользователь не сможет одновременно воспроизводить запись, если в этот момент времени он производит запись. В таком случае кнопка воспроизведения попросту не сработает. Также в этом режиме пользователь может нажать кнопку записи, при этом, ничего не записав, закончить запись. В таком случае новая запись создаваться не будет.

Также при выборе урока в режиме «Обучение» пользователь может выбрать тот урок, до которого он еще пока что не дошел. В такое случае пользователь получит сообщение об ошибке, проиллюстрированное на рисунке 4.14.

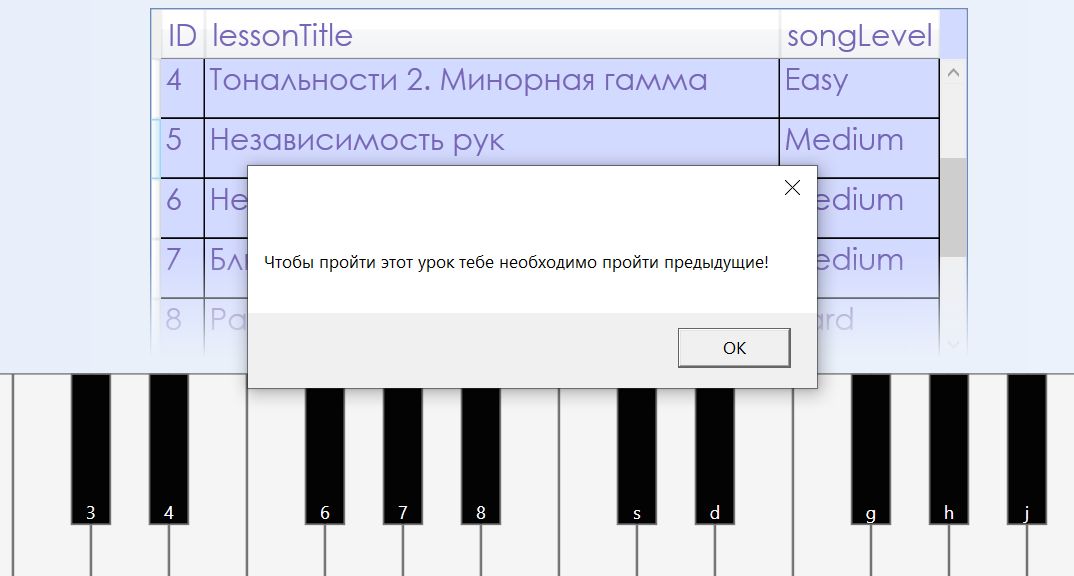


Рисунок 4.14 − Неверно выбранный урок

# Руководство по установке и использованию

## Установка программного средства

Для работы программного средства необходима первоначальная настройка. Для начала, необходимо наличие установленной на компьютере системы управления базами данных Microsoft SQL Server, а также утилиты SQL Server Management Studio для управления и администрирования компонентов Microsoft SQL Server.

Необходимо войти в SQL Server Management Studio, где в обозревателе объектов раскрыть список текущего подключения и нажать правой кнопкой мыши на вкладке «Базы данных». В контекстном меню требуется выбрать пункт «Присоединить», как на рисунке 5.1.

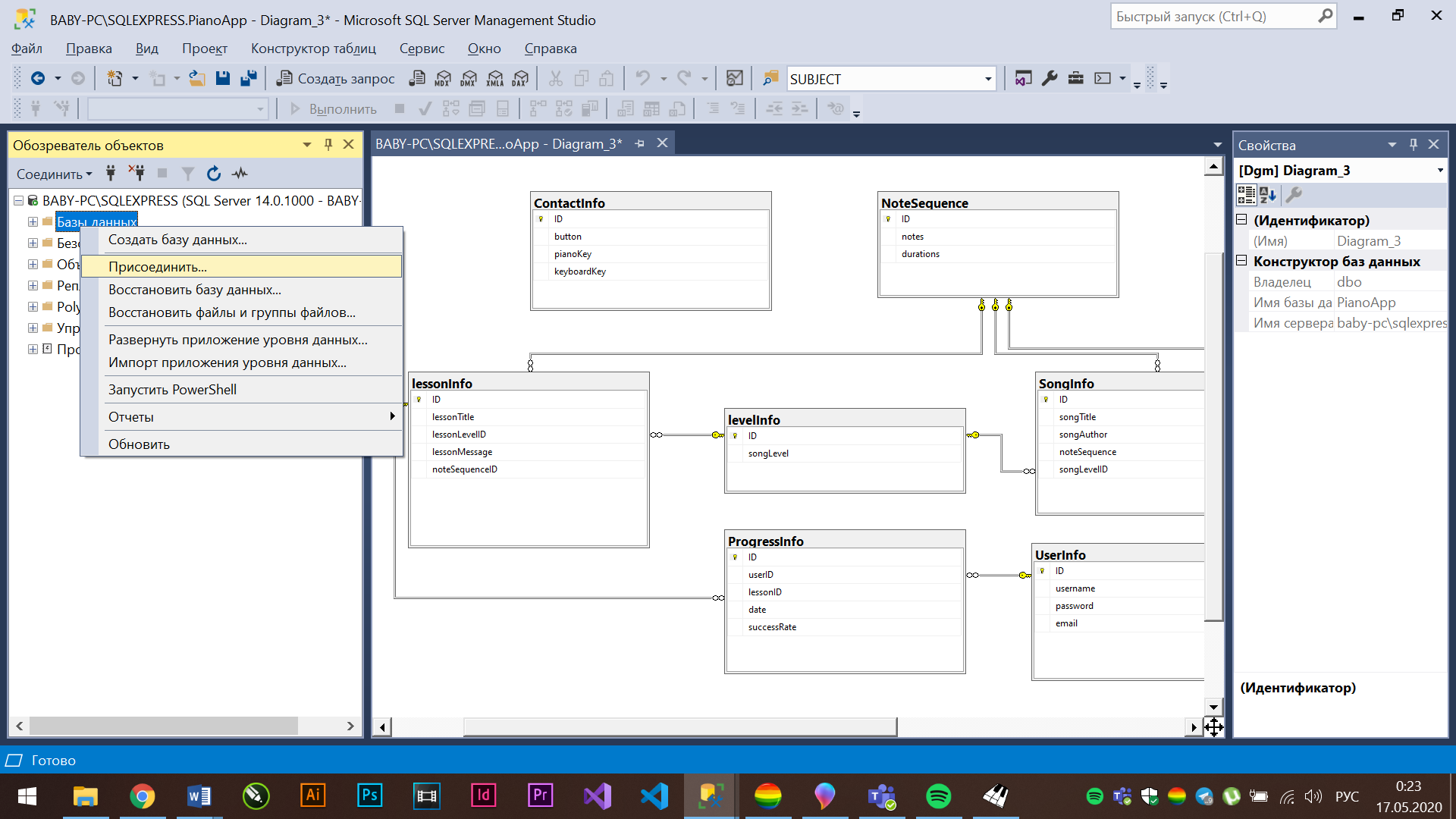


Рисунок 5.1 − Выбор пункта «Присоединить»

В открывшемся окне на панели «Базы данных для присоединения» необходимо нажать кнопку «Добавить» и указать физический путь к базе данных проекта PianoApp. После этого нажать на кнопку «ОК».

После всех проделанных шагов в раскрывающемся списке «Базы данных» обозревателя объектов должна появиться база данных PianoApp, как на рисунке 5.2.

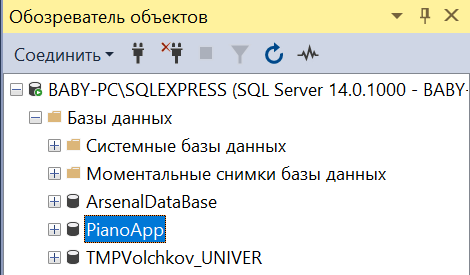


Рисунок 5.2 − Появление базы данных PianoApp

Теперь программное средство готово к запуску.

## Руководство пользователя по работе с программным средством

После успешного присоединения базы данных PianoApp пользователю необходимо запустить файл YourPiano.exe. После открытия окна авторизации новому пользователю требуется перейти к окну регистрации, в котором правильно заполнить все поля. Условия валидности вводимых данных можно прочитать, если навести курсор мыши на название поля.

Так, например, имя пользователя должно быть уникальным, т. е. пользователей с таким именем быть не должно, и содержать от 4 до 15 символов. В состав пароля должны входить латинские буквы верхнего и нижнего регистра, а также цифры, с общей длиной от 4 до 10 символов. Электронный адрес же, по условию регулярного выражения, должен состоять из четырех частей: текстовых данных до символа «@», самого символа «@», и еще двух частей после символа «@», разделенных точкой. Пример правильного заполнения полей показан на рисунке 5.3.

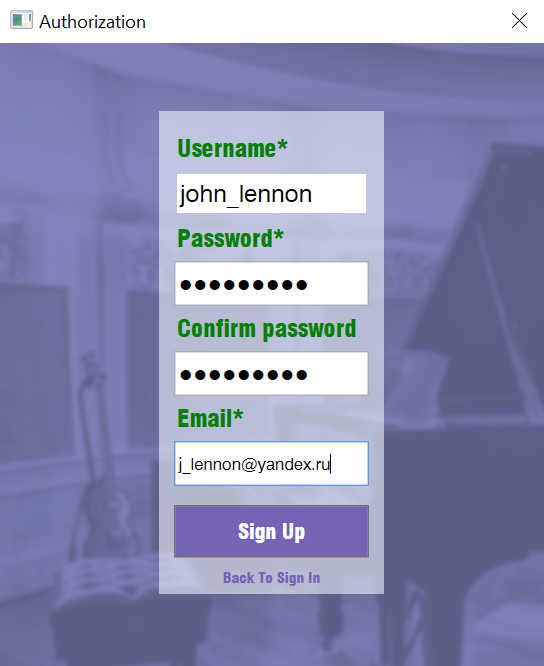


Рисунок 5.3 − Пример правильного заполнения полей окна регистрации

После успешной регистрации, пользователь авторизуется и попадает на главную страницу программного средства.

С главной страницы можно стразу попасть на страницу выбора режима игры, нажав на соответствующую кнопку.

В режиме «Свободной игры» пользователю доступна функция выбора и воспроизведения различных композиций. Для этого необходимо нажать на кнопку «SongBank» в левом верхнем углу. На экране появятся две кнопки «App songs» и «User’s songs», представленные на рисунке 5.4.

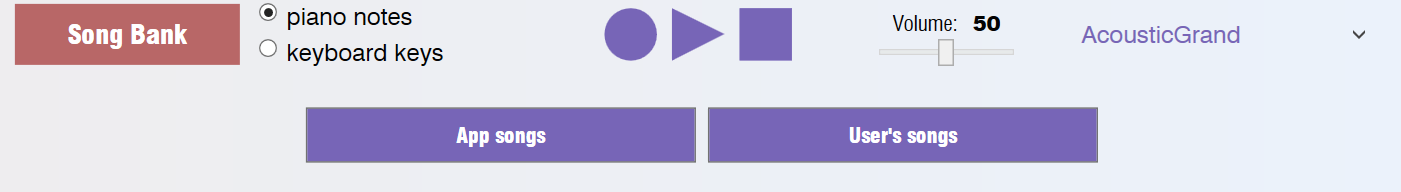


Рисунок 5.4 ­− Отображение кнопок «App songs» и «User’s songs»

Кнопка «App songs» отвечает за отображение таблицы композиций, добавленных самим разработчиком в базу данных PianoApp. Кнопка «User’s songs» отвечает за отображение композиций, записанных пользователями приложения.

После выбора необходимой композиции пользователю открывается панель, на которой отображаются текущая, предыдущая и следующая нота в выбранной композиции. Нажатие на кнопку «Play» начинает воспроизведение композиции, нажатие на кнопку «Stop» останавливает воспроизведение.

Изначально режимом отображения нот является нотная система, т. е. на экране отображаются названия самих нот. Неопытных пианистов это может поставить в тупик, поэтому режим отображения можно сменить, выбрав «keyboard keys» вместо «piano notes» на панели инструментов.

Чтобы осуществить запись собственной композиции пользователю необходимо нажать на кнопку «Record» на панели инструментов. После ее нажатия она приобретет красный цвет, как на рисунке 5.5.



Рисунок 5.5 − Включение функции записи

После того, как пользователь записал желаемую последовательность, ему необходимо повторно нажать на кнопку «Record», чтобы запись закончилась. Затем в появившемся диалоговом окне согласиться с сохранением записанного произведения. В появившемся окне ввести название произведения, как, например, на рисунке 5.6.

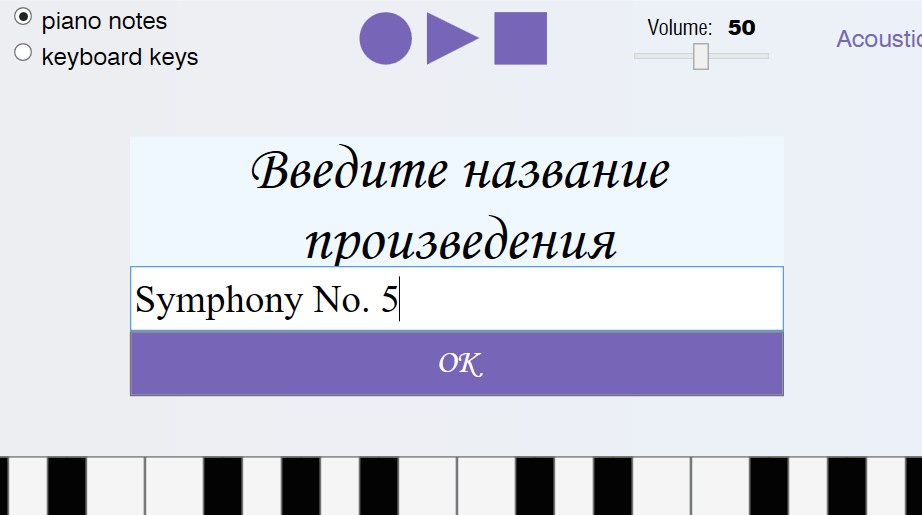


Рисунок 5.6 − Ввод названия сохраняемого произведения

Сразу после ввода названия произведение станет доступным для прослушивания, а таблица с пользовательскими композициями автоматически обновится, где можно будет найти и только что записанное произведение.

Также стоит отметить, что в данном режиме пользователь может регулировать громкость воспроизведения, а также выбирать инструмент, на котором и будет воспроизводиться композиция. Элементы управления, отвечающие за эти функции, показаны на рисунке 5.7.

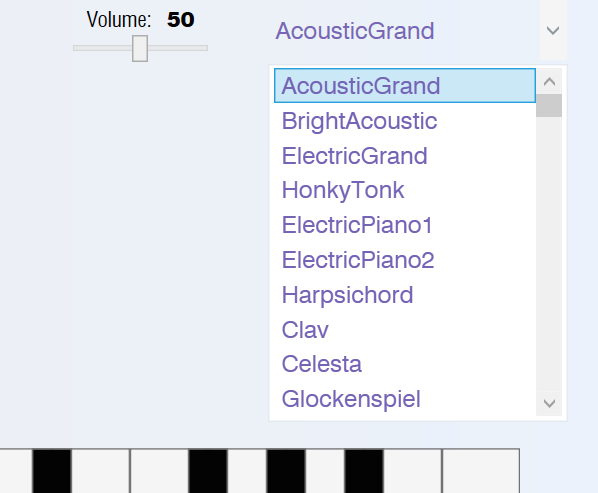


Рисунок 5.7 ­− Элементы управления регулировки громкости и смены инструмента

В режиме «Обучения» пользователю предлагается выбрать урок для его последующего прохождения из таблицы со всеми уроками. После прохождения первого урока станет доступен второй, после прохождения второго третий и т. д.

Главная отличительная особенность режима «Обучения» от режима «Свободной игры» заключается в том, что пользователю больше недоступна функция смены режима отображения нот. Также нет функции ручного регулирования воспроизведения с помощью стрелок. При возникновении трудностей пользователь может обратиться за помощью. Для этого он должен нажать на кнопку «Help!» на панели инструментов. При нажатии появляется «шпаргалка», на которой пользователь сможет узнать название интересующей его ноты. Она представлена на рисунке 5.8.

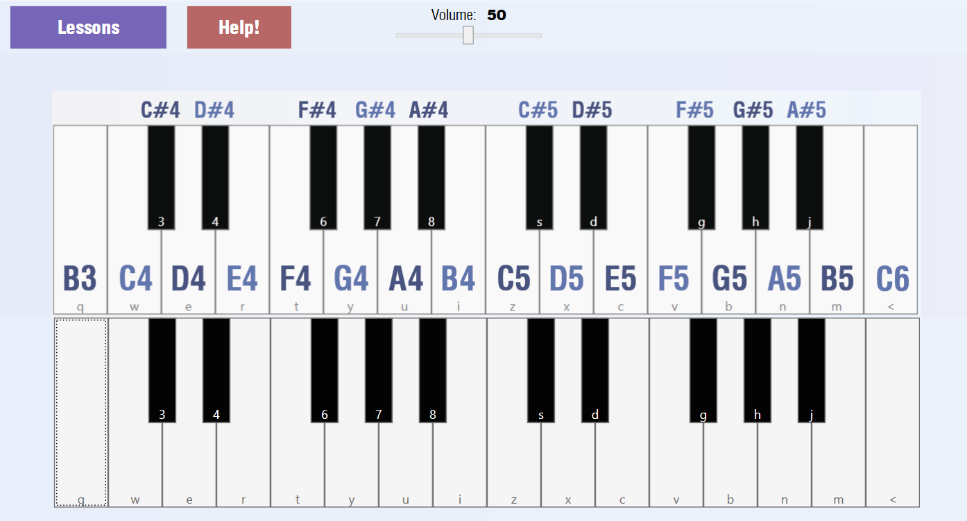


Рисунок 5.8 ­− Отображение функции помощи

Отслеживать свой прогресс пользователю позволяет страница «Прогресс», на которую он может перейти, нажав соответствующую кнопку в главном окне приложения на навигационной панели слева. Наглядное расположение кнопки показано на рисунке 5.9.

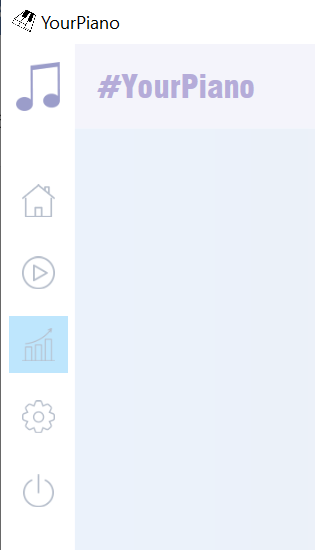


Рисунок 5.9 ­− Кнопка перехода на страницу «Прогресс»

На этой странице пользователь сможет найти всю информацию о текущем количестве пройденных уроков, среднем проценте правильно нажимаемых клавиш фортепиано, проценте пользователей, которых данный пользователь превосходит по количеству пройденных уроков, или, в случае их равенства, среднему проценту успешности прохождения уроков, а также суммарном количестве записанных композиций.

Если же пользователь захочет начать прохождение уроков заново, или вовсе полностью удалить свой профиль, для таких целей существует страница «Настройки», для открытия которой необходимо нажать кнопку, изображенную на рисунке 5.10.

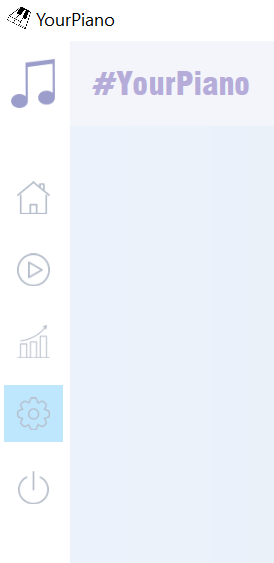


Рисунок 5.10 ­− Кнопка перехода на страницу «Настройки»

Для выхода из приложения предусмотрена кнопка «Выход», расположенная в главном окне приложения на навигационной панели слева в самом низу. Ее расположение продемонстрировано на рисунке 5.11.

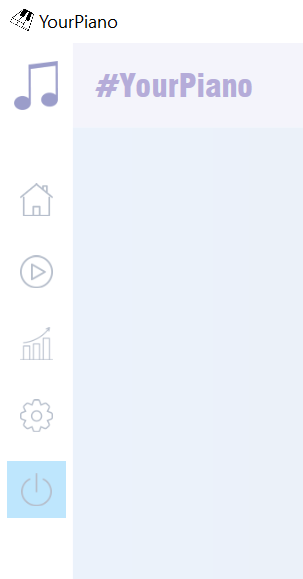


Рисунок 5.11 ­− Кнопка выхода из приложения

# Заключение

В ходе выполнения курсового проектирования было разработано и реализовано программное средство «Самоучитель игры на фортепиано». Приложение позволит любому желающему освоить базовые принципы игры на фортепиано, а также просто занимательно проводить время занимаясь творчеством.

Задачи, поставленные вначале выполнения проектирования, были выполнены. В процессе их выполнения были усвоены и закреплены новые знания в области программирования на языке C#, технологии WPF, архитектурного паттерна MVVM, а также практического применения баз данных и их проектирования в системе управления базами данных Microsoft SQL Server.

Таким образом, подводя итог, необходимо выделить ценнейший практический опыт, полученный за время работы над курсовым проектом. Полученные знания будут применяться и в дальнейшем при разработке и проектировании различных программных средств.

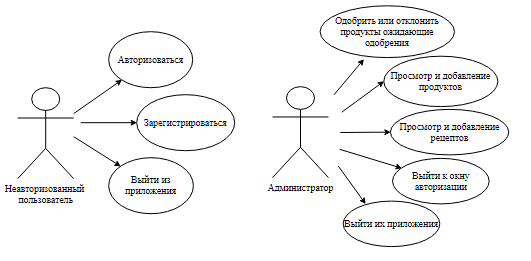
Допишите еще парочку предложений, чтобы заполнить страницу на 2/3

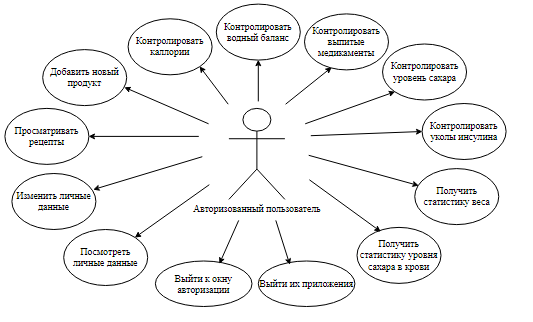
За исключением парочки недочетов, замечательная записка и как весь проект в целом ☺

# Список использованных источников

1. Everyone Piano − Forever Free Piano Software [Электронный ресурс]. − Режим доступа: https://www.everyonepiano.com/ Дата доступа: 17.05.2020
2. Dream Computer Piano 2.10 [Электронный ресурс]. − Режим доступа: https://www.malavida.com/en/soft/dream-computer-piano/ Дата доступа: 17.05.2020
3. PC 73 Virtual Piano Keyboard [Электронный ресурс]. − Режим доступа: http://www.chordpulse.com/pc-virtual-piano-keyboard.html Дата доступа: 17.05.2020
4. Garofallo, R. Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern / R. Garofallo − California: O’Reilly Media, − 2011 − 220 p.
5. Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL / К. Дж. Дейт. − СПб : Символ-Плюс, 2010. − 480 c., ил.

**Приложение А**



****

# Приложение 1. Листинг скриптов

Класс AuthorizationViewModel:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using YourPiano.Models;

namespace YourPiano.ViewModels

{

public class AuthorizationViewModel : BaseViewModel

{

private AuthorizationModel \_authorizationModel = new AuthorizationModel();

private string \_username;

public string Username

{

get => \_username;

set

{

\_username = value;

OnPropertyChanged(nameof(Username));

}

}

private string \_password;

public string Password

{

get => \_password;

set

{

\_password = value;

OnPropertyChanged(nameof(Password));

}

}

private string \_foregroundUsername = "Black";

public string ForegroundUsername

{

get => \_foregroundUsername;

set

{

\_foregroundUsername = value;

OnPropertyChanged(nameof(ForegroundUsername));

}

}

private string \_foregroundPassword = "Black";

public string ForegroundPassword

{

get => \_foregroundPassword;

set

{

\_foregroundPassword = value;

OnPropertyChanged(nameof(ForegroundPassword));

}

}

private bool \_isUsernameCorrect;

public bool IsUsernameCorrect

{

get => \_isUsernameCorrect;

set

{

\_isUsernameCorrect = value;

ForegroundUsername = \_isUsernameCorrect ? "Black" : "Red";

OnPropertyChanged(nameof(\_isUsernameCorrect));

}

}

private bool \_isPasswordCorrect;

public bool IsPasswordCorrect

{

get => \_isPasswordCorrect;

set

{

\_isPasswordCorrect = value;

ForegroundPassword = \_isPasswordCorrect ? "Black" : "Red";

OnPropertyChanged(nameof(IsPasswordCorrect));

}

}

private RelayCommand logInCommand;

public RelayCommand LogInCommand

{

get

{

return logInCommand ??

(logInCommand = new RelayCommand(obj =>

{

try

{

IsUsernameCorrect = \_authorizationModel.CheckUsername(Username);

IsPasswordCorrect = \_authorizationModel.CheckPassword(Username, Password);

\_authorizationModel.ActionAfterLogChecking(IsUsernameCorrect, IsPasswordCorrect);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}));

}

}

private RelayCommand \_openRegistrationWindow;

public RelayCommand OpenRegistrationWindow

{

get

{

return \_openRegistrationWindow ??

(\_openRegistrationWindow = new RelayCommand(obj =>

{

try

{

\_authorizationModel.OpenRegistrationWindow();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}));

}

}

public AuthorizationModel AuthorizationModel

{

get => default;

set

{

}

}

}

}

Класс AuthorizationModel

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Media;

namespace YourPiano.Models

{

public class AuthorizationModel

{

public bool CheckUsername(string currentUsername)

{

//проверка логина

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select \* from UserInfo where username='" + currentUsername + "';";

dbConn.Open();

DataTable usersDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(usersDataTable);

return usersDataTable.Rows.Count > 0 ? true : false;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return false;

}

catch (Exception ex) { MessageBox.Show(ex.Message); return false; }

finally

{

dbConn.Close();

}

}

}

public bool CheckPassword(string currentUsername, string currentPassword)

{

//проверка пароля

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select \* from UserInfo where username='" + currentUsername + "' and password ='" + currentPassword + "';";

dbConn.Open();

DataTable usersDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(usersDataTable);

CurrentUser.currentUsername = usersDataTable.Rows.Count > 0 ? currentUsername.ToLower() : "";

CurrentUser.userID = usersDataTable.Rows.Count > 0 ? Convert.ToString(usersDataTable.Rows[0][0]) : "";

return usersDataTable.Rows.Count > 0 ? true : false;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return false;

}

catch (Exception ex) { MessageBox.Show(ex.Message); return false; }

finally

{

dbConn.Close();

}

}

}

public void ActionAfterLogChecking(bool u, bool p)

{

//проерка логина и пароля для последующего открытия главного окна

//сбрасываем красную границу

foreach (Window window in App.Current.Windows)

{

if (window is Views.Authorization)

{

(window as Views.Authorization).username.BorderThickness = new Thickness(0);

(window as Views.Authorization).password.BorderThickness = new Thickness(0);

};

}

//если данные введены верно

if (u && p)

{

Views.MainWindow main = new Views.MainWindow();

main.Show();

foreach (Window window in App.Current.Windows)

{

// если окно - объект Authorization

if (window is Views.Authorization)

{

window.Close();

}

}

}

//если неверный логин

if (!u && p)

{

foreach (Window window in App.Current.Windows)

{

if (window is Views.Authorization)

{

(window as Views.Authorization).username.BorderBrush = Brushes.Red;

(window as Views.Authorization).username.BorderThickness = new Thickness(2);

};

}

}

//если неверный пароль

if (u && !p)

{

foreach (Window window in App.Current.Windows)

{

if (window is Views.Authorization)

{

(window as Views.Authorization).password.BorderBrush = Brushes.Red;

(window as Views.Authorization).password.BorderThickness = new Thickness(2);

};

}

}

//если неверный и логин и пароль

if (!u && !p)

{

foreach (Window window in App.Current.Windows)

{

if (window is Views.Authorization)

{

(window as Views.Authorization).username.BorderBrush = Brushes.Red;

(window as Views.Authorization).username.BorderThickness = new Thickness(2);

(window as Views.Authorization).password.BorderBrush = Brushes.Red;

(window as Views.Authorization).password.BorderThickness = new Thickness(2);

};

}

}

}

public void OpenRegistrationWindow()

{

Views.Registration registration = new Views.Registration();

registration.Show();

foreach (Window window in App.Current.Windows)

{

// если окно - объект Authorization

if (window is Views.Authorization)

window.Close();

}

}

}

}

Класс RegistrationViewModel

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using YourPiano.Models;

namespace YourPiano.ViewModels

{

class RegistrationViewModel : BaseViewModel

{

private RegistrationModel \_registrationModel = new RegistrationModel();

private string \_username;

public string Username

{

get => \_username;

set

{

\_username = value;

OnPropertyChanged(nameof(Username));

}

}

private string \_password;

public string Password

{

get => \_password;

set

{

\_password = value;

OnPropertyChanged(nameof(Password));

}

}

private string \_passwordConfirm;

public string PasswordConfirm

{

get => \_passwordConfirm;

set

{

\_passwordConfirm = value;

OnPropertyChanged(nameof(PasswordConfirm));

}

}

private string \_email;

public string Email

{

get => \_email;

set

{

\_email = value;

OnPropertyChanged(nameof(Email));

}

}

private string \_foregroundUsername = "Black";

public string ForegroundUsername

{

get => \_foregroundUsername;

set

{

\_foregroundUsername = value;

OnPropertyChanged(nameof(ForegroundUsername));

}

}

private string \_foregroundPassword = "Black";

public string ForegroundPassword

{

get => \_foregroundPassword;

set

{

\_foregroundPassword = value;

OnPropertyChanged(nameof(ForegroundPassword));

}

}

private string \_foregroundPasswordConfirm = "Black";

public string ForegroundPasswordConfirm

{

get => \_foregroundPasswordConfirm;

set

{

\_foregroundPasswordConfirm = value;

OnPropertyChanged(nameof(ForegroundPasswordConfirm));

}

}

private string \_foregroundEmail = "Black";

public string ForegroundEmail

{

get => \_foregroundEmail;

set

{

\_foregroundEmail = value;

OnPropertyChanged(nameof(ForegroundEmail));

}

}

private RelayCommand \_changeForegroundUsername;

public RelayCommand ChangeForegroundUsername

{

get

{

return \_changeForegroundUsername ??

(\_changeForegroundUsername = new RelayCommand(obj =>

{

ForegroundUsername = \_registrationModel.CheckUsername(\_username);

}));

}

}

private RelayCommand \_changeForegroundPassword;

public RelayCommand ChangeForegroundPassword

{

get

{

return \_changeForegroundPassword ??

(\_changeForegroundPassword = new RelayCommand(obj =>

{

foreach (Window window in App.Current.Windows)

{

if (window is Views.Registration)

{

Password = (window as Views.Registration).password.Password;

PasswordConfirm = (window as Views.Registration).passwordConfirm.Password;

};

}

ForegroundPassword = \_registrationModel.CheckPassword(\_password);

ForegroundPasswordConfirm = \_registrationModel.CheckPassword(\_password, \_passwordConfirm);

}));

}

}

private RelayCommand \_changeForegroundPasswordConfirm;

public RelayCommand ChangeForegroundPasswordConfirm

{

get

{

return \_changeForegroundPasswordConfirm ??

(\_changeForegroundPasswordConfirm = new RelayCommand(obj =>

{

foreach (Window window in App.Current.Windows)

{

if (window is Views.Registration)

{

PasswordConfirm = (window as Views.Registration).passwordConfirm.Password;

};

}

ForegroundPasswordConfirm = \_registrationModel.CheckPassword(\_password, \_passwordConfirm);

}));

}

}

private RelayCommand \_changeForegroundEmail;

public RelayCommand ChangeForegroundEmail

{

get

{

return \_changeForegroundEmail ??

(\_changeForegroundEmail = new RelayCommand(obj =>

{

ForegroundEmail = \_registrationModel.CheckEmail(\_email);

}));

}

}

private RelayCommand \_signIn;

public RelayCommand SignIn

{

get

{

return \_signIn ??

(\_signIn = new RelayCommand(obj =>

{

if (\_registrationModel.CheckInfo(\_foregroundUsername, \_foregroundPassword, \_foregroundPasswordConfirm, \_foregroundEmail))

{

\_registrationModel.NewUser(\_username, \_password, \_email);

}

else MessageBox.Show("Проверьте введенные данные");

}));

}

}

private RelayCommand \_openAuthorizationWindow;

public RelayCommand OpenAuthorizationWindow

{

get

{

return \_openAuthorizationWindow ??

(\_openAuthorizationWindow = new RelayCommand(obj =>

{

\_registrationModel.OpenAuthorizationWindow();

}));

}

}

}

}

Класс RegistrationModel

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows;

namespace YourPiano.Models

{

class RegistrationModel

{

public string CheckUsername(string currentUsername)

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select \* from UserInfo where username='" + currentUsername + "';";

dbConn.Open();

DataTable usersDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(usersDataTable);

return usersDataTable.Rows.Count == 0 && currentUsername.Length > 3 && currentUsername.Length < 15? "Green" : "Red";

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return "Red";

}

catch (Exception ex) { MessageBox.Show(ex.Message); return "Red"; }

finally

{

dbConn.Close();

}

}

}

public string CheckPassword(string currentPassword)

{

Regex regex = new Regex(@"^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z])[0-9A-Za-z]{4,10}$");

return regex.IsMatch(currentPassword) ? "Green" : "Red";

}

public string CheckPassword(string currentPassword, string currentPasswordConfirm)

{

return CheckPassword(currentPassword) == "Green" && currentPasswordConfirm == currentPassword ? "Green" : "Red";

}

public string CheckEmail(string currentEmail)

{

Regex regex = new Regex(@"^.+@.+\..+$");

return regex.IsMatch(currentEmail) ? "Green" : "Red";

}

public bool CheckInfo(string username, string password, string passwordConfirm, string email)

{

return username == "Green" &&

password == "Green" &&

passwordConfirm == "Green" &&

email == "Green" ? true : false;

}

public void NewUser(string username, string password, string email)

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "insert into UserInfo(username, password, email) " +

"values ('" + username.ToLower() + "', '" + password + "', '" + email + "');";

dbConn.Open();

SqlCommand command = new SqlCommand(querytString, dbConn);

command.ExecuteNonQuery();

MessageBox.Show("Регистрация успешна!");

OpenAuthorizationWindow();

}

catch (SqlException ex) { MessageBox.Show(ex.Message); }

catch (Exception ex) { MessageBox.Show(ex.Message); }

finally { dbConn.Close(); }

}

}

public void OpenAuthorizationWindow()

{

Views.Authorization authorization = new Views.Authorization();

authorization.Show();

foreach (Window window in App.Current.Windows)

{

// если окно - объект Registration

if (window is Views.Registration)

window.Close();

}

}

}

}

Класс PianoRollViewModel

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using YourPiano.Models;

using YourPiano.Views;

using YourPiano.Views.PianoPages;

namespace YourPiano.ViewModels

{

class PianoRollViewModel : BaseViewModel

{

public PianoRollModel \_model = new PianoRollModel();

private string \_pressedButton;

public string PressedButton

{

get => \_pressedButton;

set

{

\_pressedButton = value;

OnPropertyChanged(nameof(PressedButton));

}

}

private RelayCommand \_playKey;

public RelayCommand PlayKey

{

get

{

return \_playKey ??

(\_playKey = new RelayCommand(obj =>

{

try { \_model.PlayPianoKeyByKey(PressedButton.ToLower()); }

catch(Exception ex) { MessageBox.Show(ex.Message); }

}));

}

}

private RelayCommand \_stopKey;

public RelayCommand StopKey

{

get

{

return \_stopKey ??

(\_stopKey = new RelayCommand(obj =>

{

try { \_model.StopPianoKeyByKey(PressedButton.ToLower()); }

catch (Exception ex) { MessageBox.Show(ex.Message); }

}));

}

}

private RelayCommand \_playKeyClick;

public RelayCommand PlayKeyClick

{

get

{

return \_playKeyClick ??

(\_playKeyClick = new RelayCommand(obj =>

{

try

{

FreePlayViewModel page = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

\_model.PlayPianoKeyByKey(obj.ToString().ToLower());

page.CheckKey(obj.ToString().ToLower(), "MouseDown");

}

catch (InvalidCastException)

{

LessonsPianoViewModel page = (LessonsPianoViewModel)PageController.GetViewModel().CurrentPage.DataContext;

\_model.PlayPianoKeyByKey(obj.ToString().ToLower());

page.CheckKey(obj.ToString().ToLower(), "MouseDown");

}

catch(Exception ex) { MessageBox.Show(ex.Message); }

}));

}

}

private RelayCommand \_stopKeyClick;

public RelayCommand StopKeyClick

{

get

{

return \_stopKeyClick ??

(\_stopKeyClick = new RelayCommand(obj =>

{

try

{

FreePlayViewModel page = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

\_model.StopPianoKeyByKey(obj.ToString().ToLower());

page.CheckKey(obj.ToString().ToLower(), "MouseUp");

}

catch (InvalidCastException)

{

LessonsPianoViewModel page = (LessonsPianoViewModel)PageController.GetViewModel().CurrentPage.DataContext;

\_model.StopPianoKeyByKey(obj.ToString().ToLower());

page.CheckKey(obj.ToString().ToLower(), "MouseUp");

}

catch(Exception ex) { MessageBox.Show(ex.Message); }

}));

}

}

}

}

Класс PianoRollModel

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using Toub.Sound.Midi;

namespace YourPiano.Models

{

public class PianoRollModel

{

public byte volume;

List<string> pressedNotes = new List<string>();

public Dictionary<string, string> \_notesByKeys = new Dictionary<string, string>();

GeneralMidiInstruments generalInstrument;

public Dictionary<string, string> GetNotesByKeysFromDB()

{

Dictionary<string, string> valuePairs = new Dictionary<string, string>();

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select button, pianoKey from ContactInfo";

dbConn.Open();

DataTable notesDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(notesDataTable);

for (int i = 0; i < notesDataTable.Rows.Count; i++)

valuePairs.Add(notesDataTable.Rows[i][0].ToString().ToLower(), notesDataTable.Rows[i][1].ToString());

return valuePairs;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return valuePairs;

}

finally

{

dbConn.Close();

}

}

}

public PianoRollModel()

{

\_notesByKeys = GetNotesByKeysFromDB();

MidiPlayer.OpenMidi();//Iniciem el objecte MIDI

volume = 63;

MidiPlayer.Play(new ProgramChange(0, 0, generalInstrument));

}

public void PlayPianoKeyByKey(string key)

{

string value;

if (\_notesByKeys.TryGetValue(key, out value))

{

if (!pressedNotes.Contains(value))

{

pressedNotes.Add(value);

MidiPlayer.Play(new NoteOn(0, 0, value, volume));

}

}

}

public void PlayPianoKeyByPianoKey(string key)

{

MidiPlayer.Play(new NoteOn(0, 0, key, volume));

}

public void StopPianoKeyByKey(string key)

{

string value;

if (\_notesByKeys.TryGetValue(key, out value))

{

pressedNotes.Remove(value);

MidiPlayer.Play(new NoteOff(0, 0, value, volume));

}

}

public void StopPianoKeyByPianoKey(string key)

{

MidiPlayer.Play(new NoteOff(0, 0, key, volume));

}

public void ChangeInstrument(GeneralMidiInstruments newInstrument)

{

generalInstrument = newInstrument;

MidiPlayer.Play(new ProgramChange(0, 0, generalInstrument));

}

public void ChangeVolume(byte newVolume)

{

volume = newVolume;

}

}

}

Класс FreePlayViewModel

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Forms;

using System.Windows.Input;

using Toub.Sound.Midi;

using YourPiano.Models;

using YourPiano.Views.PianoPages;

namespace YourPiano.ViewModels

{

class FreePlayViewModel : BaseViewModel

{

PianoPlayingSongsModel \_model = new PianoPlayingSongsModel(); //model

public FreePlayPage view; //view

public FreePlayViewModel()

{

try

{

SongTable = \_model.GetSongDataBase();

SongUsersTable = \_model.GetUsersSongDataBase();

SelectedIndex = -1;

SelectedUserSongIndex = -1;

CurrentNoteLabelBackground = "#B76565";

IsSongPanelVisible = "Hidden";

IsSongListVisible = "Hidden";

PlayButtonFill = "#7563B6";

StopButtonFill = "#7563B6";

RecordButtonFill = "#7563B6";

ProgramSongsVisibility = "Hidden";

UsersSongsVisibility = "Hidden";

SongBankButtonBrush = "#7563B6";

RecordedSongTitle = "Untiteled";

Volume = 50;

notesByKeys = DataBaseInfo.GetKeysByNotes();

\_model.volume = Convert.ToByte(Convert.ToInt32(Volume \* 1.27));

RecordedSongVisibility = "Hidden";

KeyboardKeysVisibility = "Hidden";

isRecording = false;

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

}

} //viewModel constructor

List<string> pressedButtons = new List<string>(); //array of pressed keys

#region Recording Logic

Dictionary<string, string> pressedRecordingButtons = new Dictionary<string, string>();

public bool isRecording;

public string RecordedNotes;

public string RecordedDurations;

public DateTime previousDate, lastPressDate;

public void AddRecordedNote(string key)

{

if (isRecording && pressedButtons.Contains(key))

{

try

{

PianoRollModel pianoModel = PageController.GetPianoRollModel();

Dictionary<string, string> \_notesByKeys = pianoModel.\_notesByKeys;

string value, dur;

if (\_notesByKeys.TryGetValue(key, out value) && pressedRecordingButtons.TryGetValue(key, out dur))

{

RecordedNotes += value + " ";

RecordedDurations += dur + " ";

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

}

}

}

private string \_recordedSongVisibility;

public string RecordedSongVisibility

{

get => \_recordedSongVisibility;

set

{

\_recordedSongVisibility = value;

OnPropertyChanged(nameof(RecordedSongVisibility));

}

}

private string \_recordButtonFill;

public string RecordButtonFill

{

get => \_recordButtonFill;

set

{

\_recordButtonFill = value;

OnPropertyChanged(nameof(RecordButtonFill));

}

}

private string \_recordedSongTitle;

public string RecordedSongTitle

{

get => \_recordedSongTitle;

set

{

\_recordedSongTitle = value;

OnPropertyChanged(nameof(RecordedSongTitle));

}

}

private RelayCommand \_saveRecordedSong;

public RelayCommand SaveRecordedSong

{

get

{

return \_saveRecordedSong ??

(\_saveRecordedSong = new RelayCommand(obj =>

{

try

{

RecordedNotes = RecordedNotes.Substring(0, RecordedNotes.Length - 1);

RecordedDurations += Convert.ToInt32((lastPressDate - previousDate).TotalMilliseconds).ToString();

\_model.SaveSong(RecordedNotes, RecordedDurations, RecordedSongTitle);

SongName = RecordedSongTitle;

AuthorSong = CurrentUser.currentUsername + " -";

SelectedUserSongIndex = 0;

SongUsersTable.Merge(\_model.GetUsersSongDataBase());

DataTable sequenceID = DataBaseInfo.GetNoteSequenceNumber(RecordedNotes);

DataTable actionID = DataBaseInfo.GetActionNumber(sequenceID.Rows[0][0].ToString());

for (int i = 0; i < SongUsersTable.Rows.Count; i++)

if (SongUsersTable.Rows[i][0].ToString() == actionID.Rows[0][0].ToString())

SelectedUserSongIndex = i;

RecordedSongTitle = "Untitled";

if (IsSongPanelVisible == "Hidden")

IsSongPanelVisible = "Visible";

RecordedSongVisibility = "Hidden";

}

catch(Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

}

}));

}

}

private RelayCommand \_record;

public RelayCommand Record

{

get

{

return \_record ??

(\_record = new RelayCommand(obj =>

{

try

{

if (!isRecording && !isAlreadyPlaying)

{

isRecording = true;

RecordButtonFill = "#B76565";

RecordedNotes = "";

RecordedDurations = "";

previousDate = DateTime.Now;

}

else if (isRecording && !isAlreadyPlaying && RecordedNotes != "")

{

isRecording = false;

RecordButtonFill = "#7563B6";

if (System.Windows.Forms.MessageBox.Show("Сохранить запись?", "", System.Windows.Forms.MessageBoxButtons.YesNo) == DialogResult.Yes)

{

RecordedSongVisibility = "Visible";

}

}

else if (RecordedNotes == "")

{

isRecording = false;

RecordButtonFill = "#7563B6";

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

}

}));

}

}

#endregion

#region Note Navigation Logic

Dictionary<string, string> notesByKeys;

private string \_keyboardKeysVisibility;

public string KeyboardKeysVisibility

{

get => \_keyboardKeysVisibility;

set

{

\_keyboardKeysVisibility = value;

OnPropertyChanged(nameof(KeyboardKeysVisibility));

}

}

private string \_pianoKeysVisibility;

public string PianoKeysVisibility

{

get => \_pianoKeysVisibility;

set

{

\_pianoKeysVisibility = value;

OnPropertyChanged(nameof(PianoKeysVisibility));

}

}

private int \_currentNoteNumber;

public int CurrentNoteNumber

{

get => \_currentNoteNumber;

set

{

\_currentNoteNumber = value;

SetNotesOnScreen(\_currentNoteNumber);

OnPropertyChanged(nameof(CurrentNoteNumber));

}

}

private string \_previousNote;

public string PreviousNote

{

get => \_previousNote;

set

{

\_previousNote = value;

OnPropertyChanged(nameof(PreviousNote));

}

}

private string \_currentNote;

public string CurrentNote

{

get => \_currentNote;

set

{

\_currentNote = value;

OnPropertyChanged(nameof(CurrentNote));

}

}

private string \_nextNote;

public string NextNote

{

get => \_nextNote;

set

{

\_nextNote = value;

OnPropertyChanged(nameof(NextNote));

}

}

private string \_previousKeyboardNote;

public string PreviousKeyboardNote

{

get => \_previousKeyboardNote;

set

{

\_previousKeyboardNote = value;

OnPropertyChanged(nameof(PreviousKeyboardNote));

}

}

private string \_currentKeyboardNote;

public string CurrentKeyboardNote

{

get => \_currentKeyboardNote;

set

{

\_currentKeyboardNote = value;

OnPropertyChanged(nameof(CurrentKeyboardNote));

}

}

private string \_nextKeyboardNote;

public string NextKeyboardNote

{

get => \_nextKeyboardNote;

set

{

\_nextKeyboardNote = value;

OnPropertyChanged(nameof(NextKeyboardNote));

}

}

private RelayCommand \_setPreviousNote;

public RelayCommand SetPreviousNote

{

get

{

return \_setPreviousNote ??

(\_setPreviousNote = new RelayCommand(obj =>

{

\_model.PlaySingleReverseNote();

}));

}

}

private RelayCommand \_pianoNotesChecked;

public RelayCommand PianoNotesChecked

{

get

{

return \_pianoNotesChecked ??

(\_pianoNotesChecked = new RelayCommand(obj =>

{

KeyboardKeysVisibility = "Hidden";

PianoKeysVisibility = "Visible";

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_keyboardKeysChecked;

public RelayCommand KeyboardKeysChecked

{

get

{

return \_keyboardKeysChecked ??

(\_keyboardKeysChecked = new RelayCommand(obj =>

{

PianoKeysVisibility = "Hidden";

KeyboardKeysVisibility = "Visible";

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_setNextNote;

public RelayCommand SetNextNote

{

get

{

return \_setNextNote ??

(\_setNextNote = new RelayCommand(obj =>

{

\_model.PlaySingleNote();

}));

}

}

void SetNotesOnScreen(int n)

{

try

{

if (n != 0 && n != SongNotes.Length - 1)

{

CurrentNote = SongNotes[n]; PreviousNote = SongNotes[n - 1]; NextNote = SongNotes[n + 1];

CurrentKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == CurrentNote.ToLower()).Value.ToUpper();

PreviousKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == PreviousNote.ToLower()).Value.ToUpper();

NextKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == NextNote.ToLower()).Value.ToUpper();

}

if (n == 0)

{

CurrentNote = SongNotes[n]; PreviousNote = ""; NextNote = SongNotes[n + 1];

PreviousKeyboardNote = "";

CurrentKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == CurrentNote.ToLower()).Value.ToUpper();

NextKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == NextNote.ToLower()).Value.ToUpper();

}

if (n == SongNotes.Length - 1)

{

CurrentNote = SongNotes[n]; PreviousNote = SongNotes[n - 1]; NextNote = "";

CurrentKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == CurrentNote.ToLower()).Value.ToUpper();

PreviousKeyboardNote = notesByKeys.FirstOrDefault(x => x.Key == PreviousNote.ToLower()).Value.ToUpper();

NextKeyboardNote = "";

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

if (\_model.IsPlaying)

{

\_model.IsPlaying = false;

CurrentNoteNumber = 0;

\_model.CurrentNote = 0;

SetNotesOnScreen(CurrentNoteNumber);

}

}

}

#endregion

#region Current Song Info

private string \_authorSong;

public string AuthorSong

{

get => \_authorSong;

set

{

\_authorSong = value;

OnPropertyChanged(nameof(AuthorSong));

}

}

private string \_songName;

public string SongName

{

get => \_songName;

set

{

\_songName = value;

OnPropertyChanged(nameof(SongName));

}

}

private string[] \_songNotes;

public string[] SongNotes

{

get => \_songNotes;

set

{

\_songNotes = value;

SetNotesOnScreen(0);

MaxBarValue = \_songNotes.Length - 1;

OnPropertyChanged(nameof(SongNotes));

}

}

private string \_isSongPanelVisible;

public string IsSongPanelVisible

{

get => \_isSongPanelVisible;

set

{

\_isSongPanelVisible = value;

OnPropertyChanged(nameof(IsSongPanelVisible));

}

}

private string \_isSongListVisible;

public string IsSongListVisible

{

get => \_isSongListVisible;

set

{

\_isSongListVisible = value;

OnPropertyChanged(nameof(IsSongListVisible));

}

}

#endregion

#region App SongsTable Logic

private DataTable \_songTable;

public DataTable SongTable

{

get => \_songTable;

set

{

\_songTable = value;

OnPropertyChanged(nameof(SongTable));

}

}

private RelayCommand \_openSongList;

public RelayCommand OpenSongList

{

get

{

return \_openSongList ??

(\_openSongList = new RelayCommand(obj =>

{

if (!\_model.IsPlaying)

{

if (IsSongListVisible == "Hidden") { IsSongListVisible = "Visible"; SongBankButtonBrush = "#B76565"; IsSongPanelVisible = "Hidden"; }

else if (IsSongListVisible == "Visible") { IsSongListVisible = "Hidden"; SongBankButtonBrush = "#7563B6"; IsSongPanelVisible = "Visible"; }

}

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_openSong;

public RelayCommand OpenSong

{

get

{

return \_openSong ??

(\_openSong = new RelayCommand(obj =>

{

try

{

if (!\_model.IsPlaying)

{

IsSongListVisible = "Hidden";

IsSongPanelVisible = "Visible";

\_model.GetSong((int)SongTable.Rows[SelectedIndex][0]);

AuthorSong = \_model.AUTHOR + " -";

SongName = \_model.TITLE;

SongNotes = \_model.NOTES;

CurrentNoteNumber = 0;

view.pianoRoll.buttonB2.Focus();

}

}

catch(Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

}

finally { if (SongBankButtonBrush == "#B76565") SongBankButtonBrush = "#7563B6"; }

}));

}

}

private RelayCommand \_showSongs;

public RelayCommand ShowSongs

{

get

{

return \_showSongs ??

(\_showSongs = new RelayCommand(obj =>

{

if (UsersSongsVisibility == "Visible") UsersSongsVisibility = "Hidden";

if (ProgramSongsVisibility == "Hidden") ProgramSongsVisibility = "Visible";

}));

}

}

private int \_selectedIndex;

public int SelectedIndex

{

get => \_selectedIndex;

set

{

\_selectedIndex = value;

OnPropertyChanged(nameof(SelectedIndex));

}

}

private string \_programSongsVisibility;

public string ProgramSongsVisibility

{

get => \_programSongsVisibility;

set

{

\_programSongsVisibility = value;

OnPropertyChanged(nameof(ProgramSongsVisibility));

}

}

#endregion

#region User's SongsTable Logic

private DataTable \_songUsersTable;

public DataTable SongUsersTable

{

get => \_songUsersTable;

set

{

\_songUsersTable = value;

OnPropertyChanged(nameof(SongUsersTable));

}

}

private RelayCommand \_openUserSong;

public RelayCommand OpenUserSong

{

get

{

return \_openUserSong ??

(\_openUserSong = new RelayCommand(obj =>

{

try

{

if (!isAlreadyPlaying)

{

IsSongListVisible = "Hidden";

IsSongPanelVisible = "Visible";

\_model.GetUserSong((int)SongUsersTable.Rows[SelectedUserSongIndex][0]);

AuthorSong = \_model.AUTHOR + " -";

SongName = \_model.TITLE;

SongNotes = \_model.NOTES;

CurrentNoteNumber = 0;

view.pianoRoll.buttonB2.Focus();

}

}

catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

finally { if (SongBankButtonBrush == "#B76565") SongBankButtonBrush = "#7563B6"; }

}));

}

}

private RelayCommand \_showUsersSongs;

public RelayCommand ShowUsersSongs

{

get

{

return \_showUsersSongs ??

(\_showUsersSongs = new RelayCommand(obj =>

{

if (UsersSongsVisibility == "Hidden") UsersSongsVisibility = "Visible";

if (ProgramSongsVisibility == "Visible") ProgramSongsVisibility = "Hidden";

}));

}

}

private int \_selectedUserSongIndex;

public int SelectedUserSongIndex

{

get => \_selectedUserSongIndex;

set

{

\_selectedUserSongIndex = value;

OnPropertyChanged(nameof(SelectedUserSongIndex));

}

}

private string \_usersSongsVisibility;

public string UsersSongsVisibility

{

get => \_usersSongsVisibility;

set

{

\_usersSongsVisibility = value;

OnPropertyChanged(nameof(UsersSongsVisibility));

}

}

#endregion

#region Play/Stop Logic

bool isAlreadyPlaying;

private RelayCommand \_playSong;

public RelayCommand PlaySong

{

get

{

return \_playSong ??

(\_playSong = new RelayCommand(obj =>

{

try

{

if (IsSongPanelVisible == "Visible" && IsSongListVisible == "Hidden")

{

if (!isAlreadyPlaying && !isRecording)

{

PlayButtonFill = "#B76565";

isAlreadyPlaying = true;

\_model.PlayNOTES();

}

}

}

catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

finally { view.pianoRoll.buttonB2.Focus(); }

}));

}

}

private RelayCommand \_stopSong;

public RelayCommand StopSong

{

get

{

return \_stopSong ??

(\_stopSong = new RelayCommand(obj =>

{

try

{

if (isAlreadyPlaying)

{

isAlreadyPlaying = false;

PlayButtonFill = "#7563B6";

\_model.IsPlaying = false;

view.pianoRoll.buttonB2.Focus();

}

}

catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

}));

}

}

private string \_stopButtonFill;

public string StopButtonFill

{

get => \_stopButtonFill;

set

{

\_stopButtonFill = value;

OnPropertyChanged(nameof(StopButtonFill));

}

}

private string \_playButtonFill;

public string PlayButtonFill

{

get => \_playButtonFill;

set

{

\_playButtonFill = value;

OnPropertyChanged(nameof(PlayButtonFill));

}

}

#endregion

#region Volume/InstrumentChange Logic

private int \_volume;

public int Volume

{

get => \_volume;

set

{

\_volume = value;

OnPropertyChanged(nameof(Volume));

}

}

private RelayCommand \_changeVolume;

public RelayCommand ChangeVolume

{

get

{

return \_changeVolume ??

(\_changeVolume = new RelayCommand(obj =>

{

try { \_model.ChangeVolume(Convert.ToByte(Convert.ToInt32(Volume \* 1.27))); }

catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

finally { view.pianoRoll.buttonB2.Focus(); }

}));

}

}

private string \_selectedInstrument;

public string SelectedInstrument

{

get => \_selectedInstrument;

set

{

\_selectedInstrument = value;

OnPropertyChanged(nameof(SelectedInstrument));

}

}

private RelayCommand \_selectionInstrumentChanged;

public RelayCommand SelectionInstrumentChanged

{

get

{

return \_selectionInstrumentChanged ??

(\_selectionInstrumentChanged = new RelayCommand(obj =>

{

try { \_model.ChangeInstrument((GeneralMidiInstruments)System.Enum.Parse(typeof(GeneralMidiInstruments), SelectedInstrument)); }

catch(Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

finally { view.pianoRoll.buttonB2.Focus(); }

}));

}

}

#endregion

#region Visibilities/Colors/Others Properties

private int \_maxBarValue;

public int MaxBarValue

{

get => \_maxBarValue;

set

{

\_maxBarValue = value;

OnPropertyChanged(nameof(MaxBarValue));

}

}

private string \_currentNoteLabelBackground;

public string CurrentNoteLabelBackground

{

get => \_currentNoteLabelBackground;

set

{

\_currentNoteLabelBackground = value;

OnPropertyChanged(nameof(CurrentNoteLabelBackground));

}

}

private string \_songBankButtonBrush;

public string SongBankButtonBrush

{

get => \_songBankButtonBrush;

set

{

\_songBankButtonBrush = value;

OnPropertyChanged(nameof(SongBankButtonBrush));

}

}

#endregion

//method check pressed key on keyboard

//change piano key color

//add pressed and recorded buttons

public async void CheckKey(string key, string eventName)

{

try

{

PianoRollModel pianoModel = PageController.GetPianoRollModel();

Dictionary<string, string> \_notesByKeys = pianoModel.\_notesByKeys;

string value = "";

if (eventName == "KeyDown" || eventName == "MouseDown")

{

if (!pressedButtons.Contains(key))

{

pressedButtons.Add(key);

if (isRecording) pressedRecordingButtons.Add(key, Convert.ToInt32((DateTime.Now - previousDate).TotalMilliseconds).ToString());

previousDate = DateTime.Now;

if (\_notesByKeys.TryGetValue(key, out value))

{

if (value == CurrentNote)

{

if (CurrentNoteNumber == SongNotes.Length - 1)

{

CurrentNoteLabelBackground = "#4A982F";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

CurrentNoteLabelBackground = "#B76565";

if (!\_model.IsPlaying)

{

\_model.CurrentNote = 0;

CurrentNoteNumber = 0;

}

}

else

{

CurrentNoteLabelBackground = "#4A982F";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

CurrentNoteLabelBackground = "#B76565";

if (!\_model.IsPlaying)

{

\_model.CurrentNote += 1;

CurrentNoteNumber += 1;

}

}

}

else

{

CurrentNoteLabelBackground = "#FF4A4A";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

CurrentNoteLabelBackground = "#B76565";

}

}

}

}

if (eventName == "KeyUp" || eventName == "MouseUp")

{

lastPressDate = DateTime.Now;

AddRecordedNote(key);

pressedButtons.Remove(key);

pressedRecordingButtons.Remove(key);

}

}

catch(Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

}

}

}

Класс LessonsPianoViewModel

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using YourPiano.Models;

using YourPiano.Views.PianoPages;

namespace YourPiano.ViewModels

{

class LessonsPianoViewModel : BaseViewModel

{

public LessonsPianoViewModel()

{

try

{

LessonsTable = \_model.GetLessonsDataBase();

CurrentNoteLabelBackground = "#B76565";

HelpButtonBrush = "#7563B6";

LessonButtonBrush = "#B76565";

EndMessageVisibility = "Hidden";

LessonsTableVisibility = "Visible";

LessonPanelVisibility = "Hidden";

HelpRollVisibility = "Hidden";

MessageVisibility = "Hidden";

SelectedIndex = -1;

PressedKeysNumber = 0;

Volume = 50;

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

} //constructor

PianoPlayingSongsModel \_model = new PianoPlayingSongsModel(); //model

public LessonsPianoPage view; //LessonPage view

int PressedKeysNumber; //number of pressed keys to calculate success rate

List<string> pressedButtons = new List<string>();

#region Piano notes Navigation

private int \_currentNoteNumber;

public int CurrentNoteNumber

{

get => \_currentNoteNumber;

set

{

\_currentNoteNumber = value;

SetNotesOnScreen(\_currentNoteNumber);

OnPropertyChanged(nameof(CurrentNoteNumber));

}

}

private string \_previousNote;

public string PreviousNote

{

get => \_previousNote;

set

{

\_previousNote = value;

OnPropertyChanged(nameof(PreviousNote));

}

}

private string \_currentNote;

public string CurrentNote

{

get => \_currentNote;

set

{

\_currentNote = value;

OnPropertyChanged(nameof(CurrentNote));

}

}

private string \_nextNote;

public string NextNote

{

get => \_nextNote;

set

{

\_nextNote = value;

OnPropertyChanged(nameof(NextNote));

}

}

void SetNotesOnScreen(int n)

{

try

{

if (n != 0 && n != LessonNotes.Length - 1)

{

CurrentNote = LessonNotes[n]; PreviousNote = LessonNotes[n - 1]; NextNote = LessonNotes[n + 1];

}

if (n == 0)

{

CurrentNote = LessonNotes[n]; PreviousNote = ""; NextNote = LessonNotes[n + 1];

}

if (n == LessonNotes.Length - 1)

{

CurrentNote = LessonNotes[n]; PreviousNote = LessonNotes[n - 1]; NextNote = "";

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.Message);

if (\_model.IsPlaying)

{

\_model.IsPlaying = false;

CurrentNoteNumber = 0;

\_model.CurrentNote = 0;

SetNotesOnScreen(CurrentNoteNumber);

}

}

}

#endregion

#region Lessons Navigation

private RelayCommand \_tryAgain;

public RelayCommand TryAgain

{

get

{

return \_tryAgain ??

(\_tryAgain = new RelayCommand(obj =>

{

\_model.CurrentNote = 0;

CurrentNoteNumber = 0;

EndMessageVisibility = "Hidden";

LessonPanelVisibility = "Visible";

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_nextLesson;

public RelayCommand NextLesson

{

get

{

return \_nextLesson ??

(\_nextLesson = new RelayCommand(obj =>

{

if (SelectedIndex == LessonsTable.Rows.Count - 1)

MessageBox.Show("Это был последний урок");

else SelectedIndex++;

}));

}

}

#endregion

#region Lesson Table Logic

private DataTable \_lessonsTable;

public DataTable LessonsTable

{

get => \_lessonsTable;

set

{

\_lessonsTable = value;

OnPropertyChanged(nameof(LessonsTable));

}

}

private string \_lessonsTableVisibility;

public string LessonsTableVisibility

{

get => \_lessonsTableVisibility;

set

{

\_lessonsTableVisibility = value;

OnPropertyChanged(nameof(LessonsTableVisibility));

}

}

private int \_selectedIndex;

public int SelectedIndex

{

get => \_selectedIndex;

set

{

\_selectedIndex = value;

OnPropertyChanged(nameof(SelectedIndex));

}

}

private RelayCommand \_openLessonsTable;

public RelayCommand OpenLessonsTable

{

get

{

return \_openLessonsTable ??

(\_openLessonsTable = new RelayCommand(obj =>

{

if (EndMessageVisibility == "Hidden" && MessageVisibility == "Hidden")

{

if (LessonsTableVisibility == "Hidden") { LessonsTableVisibility = "Visible"; LessonButtonBrush = "#B76565"; LessonPanelVisibility = "Hidden"; }

else if (LessonsTableVisibility == "Visible") { LessonsTableVisibility = "Hidden"; LessonButtonBrush = "#7563B6"; LessonPanelVisibility = "Visible"; }

}

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_openLesson;

public RelayCommand OpenLesson

{

get

{

return \_openLesson ??

(\_openLesson = new RelayCommand(obj =>

{

try

{

if ((int)LessonsTable.Rows[SelectedIndex][0] <= DataBaseInfo.GetMaxPassedLevel(CurrentUser.currentUsername) + 1)

{

EndMessageVisibility = "Hidden";

LessonPanelVisibility = "Hidden";

LessonsTableVisibility = "Hidden";

\_model.GetLesson((int)LessonsTable.Rows[SelectedIndex][0]);

LessonNotes = \_model.NOTES;

CurrentNoteNumber = 0;

LessonMessage = \_model.MESSAGE;

MessageVisibility = "Visible";

}

else MessageBox.Show("Чтобы пройти этот урок тебе необходимо пройти предыдущие!");

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}));

}

}

#endregion

#region Current Lesson Info

private string[] \_lessonNotes;

public string[] LessonNotes

{

get => \_lessonNotes;

set

{

\_lessonNotes = value;

SetNotesOnScreen(0);

MaxBarValue = \_lessonNotes.Length - 1;

OnPropertyChanged(nameof(LessonNotes));

}

}

private int \_maxBarValue;

public int MaxBarValue

{

get => \_maxBarValue;

set

{

\_maxBarValue = value;

OnPropertyChanged(nameof(MaxBarValue));

}

}

private string \_currentNoteLabelBackground;

public string CurrentNoteLabelBackground

{

get => \_currentNoteLabelBackground;

set

{

\_currentNoteLabelBackground = value;

OnPropertyChanged(nameof(CurrentNoteLabelBackground));

}

}

private string \_lessonMessage;

public string LessonMessage

{

get => \_lessonMessage;

set

{

\_lessonMessage = value;

OnPropertyChanged(nameof(LessonMessage));

}

}

private string \_lessonEndMessage;

public string LessonEndMessage

{

get => \_lessonEndMessage;

set

{

\_lessonEndMessage = value;

OnPropertyChanged(nameof(LessonEndMessage));

}

}

private string \_lessonTitle;

public string LessonTitle

{

get => \_lessonTitle;

set

{

\_lessonTitle = value;

OnPropertyChanged(nameof(LessonTitle));

}

}

#endregion

#region Visibilities/Volume/Others Properties

private string \_lessonPanelVisibility;

public string LessonPanelVisibility

{

get => \_lessonPanelVisibility;

set

{

\_lessonPanelVisibility = value;

OnPropertyChanged(nameof(LessonPanelVisibility));

}

}

private string \_helpRollVisibility;

public string HelpRollVisibility

{

get => \_helpRollVisibility;

set

{

\_helpRollVisibility = value;

OnPropertyChanged(nameof(HelpRollVisibility));

}

}

private string \_messageVisibility;

public string MessageVisibility

{

get => \_messageVisibility;

set

{

\_messageVisibility = value;

OnPropertyChanged(nameof(MessageVisibility));

}

}

private string \_endMessageVisibility;

public string EndMessageVisibility

{

get => \_endMessageVisibility;

set

{

\_endMessageVisibility = value;

OnPropertyChanged(nameof(EndMessageVisibility));

}

}

private int \_volume;

public int Volume

{

get => \_volume;

set

{

\_volume = value;

OnPropertyChanged(nameof(Volume));

}

}

private RelayCommand \_showHelpImage;

public RelayCommand ShowHelpImage

{

get

{

return \_showHelpImage ??

(\_showHelpImage = new RelayCommand(obj =>

{

if (HelpRollVisibility == "Hidden") { HelpRollVisibility = "Visible"; HelpButtonBrush = "#B76565"; }

else if (HelpRollVisibility == "Visible") { HelpRollVisibility = "Hidden"; HelpButtonBrush = "#7563B6"; }

view.pianoRoll.buttonB2.Focus();

}));

}

}

private string \_lessonButtonBrush;

public string LessonButtonBrush

{

get => \_lessonButtonBrush;

set

{

\_lessonButtonBrush = value;

OnPropertyChanged(nameof(LessonButtonBrush));

}

}

private string \_helpButtonBrush;

public string HelpButtonBrush

{

get => \_helpButtonBrush;

set

{

\_helpButtonBrush = value;

OnPropertyChanged(nameof(HelpButtonBrush));

}

}

private RelayCommand \_changeVolume;

public RelayCommand ChangeVolume

{

get

{

return \_changeVolume ??

(\_changeVolume = new RelayCommand(obj =>

{

\_model.ChangeLessonVolume(Convert.ToByte(Convert.ToInt32(Volume \* 1.27)));

view.pianoRoll.buttonB2.Focus();

}));

}

}

private RelayCommand \_closeMessage;

public RelayCommand CloseMessage

{

get

{

return \_closeMessage ??

(\_closeMessage = new RelayCommand(obj =>

{

MessageVisibility = "Hidden";

LessonTitle = \_model.TITLE;

LessonPanelVisibility = "Visible";

if (LessonButtonBrush == "#B76565") LessonButtonBrush = "#7563B6";

view.pianoRoll.buttonB2.Focus();

}));

}

}

#endregion

//method check pressed keyboard key

//change piano key color

public async void CheckKey(string key, string eventName)

{

try

{

PianoRollModel pianoModel = PageController.GetLessonPianoRollModel();

Dictionary<string, string> \_notesByKeys = pianoModel.\_notesByKeys;

string value = "";

if (eventName == "KeyDown" || eventName == "MouseDown")

{

if (!pressedButtons.Contains(key))

{

pressedButtons.Add(key);

if (\_notesByKeys.TryGetValue(key, out value))

{

if (value == CurrentNote)

{

if (CurrentNoteNumber == \_lessonNotes.Length - 1)

{

PressedKeysNumber++;

CurrentNoteLabelBackground = "#4A982F";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

double successRate = Math.Round((double)LessonNotes.Length / (double)PressedKeysNumber \* 100);

LessonEndMessage = $"Поздравляем! Урок №{(int)LessonsTable.Rows[SelectedIndex][0]} успешно окончен! Процент правильных нажатий: {Math.Round(successRate, 2)}%";

\_model.SaveResults(CurrentUser.userID, LessonsTable.Rows[SelectedIndex][0].ToString(), Convert.ToString(successRate));

EndMessageVisibility = "Visible";

LessonPanelVisibility = "Hidden";

CurrentNoteLabelBackground = "#B76565";

PressedKeysNumber = 0;

}

else

{

PressedKeysNumber++;

CurrentNoteLabelBackground = "#4A982F";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

CurrentNoteLabelBackground = "#B76565";

if (!\_model.IsPlaying)

{

\_model.CurrentNote += 1;

CurrentNoteNumber += 1;

}

}

}

else

{

CurrentNoteLabelBackground = "#FF4A4A";

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

PressedKeysNumber++;

CurrentNoteLabelBackground = "#B76565";

}

}

}

}

if (eventName == "KeyUp" || eventName == "MouseUp")

{

pressedButtons.Remove(key);

}

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

}

}

Класс PianoPlayingSongsModel

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Threading;

using Toub.Sound.Midi;

using YourPiano.ViewModels;

using YourPiano.Views;

using YourPiano.Views.PianoPages;

namespace YourPiano.Models

{

class PianoPlayingSongsModel

{

const int canal = 1;

public string TITLE;

public string AUTHOR;

public string MESSAGE;

public string[] NOTES;

string[] DURATIONS;

public bool IsPlaying;

public int CurrentNote = 0;

public byte volume;

public async void PlayNOTES()

{

try

{

IsPlaying = true;

FreePlayViewModel viewModel = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

while (IsPlaying)

{

volume = PageController.GetPianoRollModel().volume; //получаем текущее значение громкости

SetPressedButtonColor(); //меняем цвет кнопки

MidiPlayer.Play(new NoteOn(0, 0, NOTES[CurrentNote], volume));

await Task.Factory.StartNew(() =>

{

try { Thread.Sleep(Convert.ToInt32(DURATIONS[CurrentNote])); }

catch { Thread.Sleep(250); }

});

SetOriginalButtonColor();//возвращаем цвет

MidiPlayer.Play(new NoteOff(0, 0, NOTES[CurrentNote], volume));

CurrentNote = CurrentNote == NOTES.Length - 1 ? 0 : CurrentNote += 1;

viewModel.CurrentNoteNumber = CurrentNote;

}

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

public DataTable GetSongDataBase()

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select SongInfo.ID, songAuthor, songTitle, songLevel from SongInfo join levelInfo on SongInfo.songLevelID = levelInfo.ID";

dbConn.Open();

DataTable songsDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(songsDataTable);

return songsDataTable;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return new DataTable();

}

catch (Exception ex) { MessageBox.Show(ex.Message); return new DataTable(); }

finally

{

dbConn.Close();

}

}

}

public DataTable GetUsersSongDataBase()

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select ActionInfo.ID, username, ActionInfo.songName, date from ActionInfo join UserInfo on ActionInfo.userID = UserInfo.ID";

dbConn.Open();

DataTable songsDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(songsDataTable);

songsDataTable.PrimaryKey = new DataColumn[] { songsDataTable.Columns["ID"] };

return songsDataTable;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return new DataTable();

}

catch(Exception ex) { MessageBox.Show(ex.Message); return new DataTable(); }

finally

{

dbConn.Close();

}

}

}

public DataTable GetLessonsDataBase()

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select lessonInfo.ID, lessonTitle, songLevel from lessonInfo join levelInfo on lessonInfo.lessonLevelID = levelInfo.ID";

dbConn.Open();

DataTable songsDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(songsDataTable);

return songsDataTable;

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

return new DataTable();

}

catch (Exception ex) { MessageBox.Show(ex.Message); return new DataTable(); }

finally

{

dbConn.Close();

}

}

}

void SetPressedButtonColor()

{

try

{

List<FrameworkElement> lstElement;

FreePlayPage page = (FreePlayPage)PageController.GetViewModel().CurrentPage;

FreePlayViewModel viewModel = (FreePlayViewModel)page.DataContext;

Panel pianoRoll = (Panel)page.pianoRoll.Content;

UIElementCollection element = pianoRoll.Children;

lstElement = element.Cast<FrameworkElement>().ToList();

Dictionary<string, string> \_notesByKeys = PageController.GetPianoRollModel().\_notesByKeys;

List<string> pressedNotes = new List<string>();

var lstControl = lstElement.OfType<Button>();

foreach (var control in lstControl)

{

if (control.Tag.ToString() == \_notesByKeys.FirstOrDefault(x => x.Value == NOTES[CurrentNote]).Key)

{

if (NOTES[CurrentNote].Contains("#"))

control.SetResourceReference(Button.StyleProperty, "BlackPressedPianoKeyStyle");

else control.SetResourceReference(Button.StyleProperty, "WhitePressedPianoKeyStyle");

}

}

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

void SetOriginalButtonColor()

{

try

{

List<FrameworkElement> lstElement;

FreePlayPage page = (FreePlayPage)PageController.GetViewModel().CurrentPage;

Panel pianoRoll = (Panel)page.pianoRoll.Content;

UIElementCollection element = pianoRoll.Children;

lstElement = element.Cast<FrameworkElement>().ToList();

Dictionary<string, string> \_notesByKeys = PageController.GetPianoRollModel().\_notesByKeys;

List<string> pressedNotes = new List<string>();

var lstControl = lstElement.OfType<Button>();

foreach (var control in lstControl)

{

if (control.Tag.ToString() == \_notesByKeys.FirstOrDefault(x => x.Value == NOTES[CurrentNote]).Key)

{

if (NOTES[CurrentNote].Contains("#"))

control.SetResourceReference(Button.StyleProperty, "BlackPianoKeyStyle");

else control.SetResourceReference(Button.StyleProperty, "WhitePianoKeyStyle");

}

}

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

public void GetSong(int IDsong)

{

CurrentNote = 0;

string notes = "";

string durations = "";

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select songAuthor, songTitle, notes, durations from SongInfo join NoteSequence on SongInfo.noteSequence = NoteSequence.ID where SongInfo.ID =" + IDsong;

dbConn.Open();

DataTable notesDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(notesDataTable);

AUTHOR = Convert.ToString(notesDataTable.Rows[0][0]);

TITLE = Convert.ToString(notesDataTable.Rows[0][1]);

notes = Convert.ToString(notesDataTable.Rows[0][2]);

durations = Convert.ToString(notesDataTable.Rows[0][3]);

NOTES = notes.Split(new char[] { ' ' });

DURATIONS = durations.Split(new char[] { ' ' });

dbConn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

}

public void GetUserSong(int IDsong)

{

CurrentNote = 0;

string notes = "";

string durations = "";

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = $"select username, songName, notes, durations from ActionInfo join UserInfo on ActionInfo.userID = UserInfo.ID join NoteSequence on ActionInfo.noteSecuence = NoteSequence.ID where ActionInfo.ID={IDsong}";

dbConn.Open();

DataTable notesDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(notesDataTable);

AUTHOR = Convert.ToString(notesDataTable.Rows[0][0]);

TITLE = Convert.ToString(notesDataTable.Rows[0][1]);

notes = Convert.ToString(notesDataTable.Rows[0][2]);

durations = Convert.ToString(notesDataTable.Rows[0][3]);

NOTES = notes.Split(new char[] { ' ' });

DURATIONS = durations.Split(new char[] { ' ' });

dbConn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

}

public void GetLesson(int IDlesson)

{

CurrentNote = 0;

string notes = "";

string durations = "";

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = "select lessonTitle, lessonMessage, notes, durations from lessonInfo join NoteSequence on lessonInfo.noteSequenceID = NoteSequence.ID where lessonInfo.ID =" + IDlesson;

dbConn.Open();

DataTable notesDataTable = new DataTable();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(notesDataTable);

TITLE = Convert.ToString(notesDataTable.Rows[0][0]);

MESSAGE = Convert.ToString(notesDataTable.Rows[0][1]);

notes = Convert.ToString(notesDataTable.Rows[0][2]);

durations = Convert.ToString(notesDataTable.Rows[0][3]);

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

finally

{

NOTES = notes.Split(new char[] { ' ' });

DURATIONS = durations.Split(new char[] { ' ' });

dbConn.Close();

}

}

}

public void SaveSong(string NewNotes, string NewDurations, string SongTitle)

{

FreePlayViewModel viewModel = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

using (SqlConnection dbConn = new SqlConnection())

{

try

{

int pos = NewDurations.IndexOf(' ');

NewDurations = NewDurations.Substring(pos+1);

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = $"insert NoteSequence(notes, durations) values ('{NewNotes}', '{NewDurations}')";

dbConn.Open();

SqlCommand command = new SqlCommand(querytString, dbConn);

command.ExecuteNonQuery();

dbConn.Close();

querytString = $"select ID from NoteSequence where notes = '{NewNotes}'";

dbConn.Open();

SqlDataAdapter dbAdapter = new SqlDataAdapter(querytString, dbConn);

DataTable id = new DataTable();

dbAdapter = new SqlDataAdapter(querytString, dbConn);

dbAdapter.Fill(id);

dbConn.Close();

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

querytString = $"insert into ActionInfo(userID, songName, date, noteSecuence)" +

$"values ('{CurrentUser.userID}', '{SongTitle}', '{DateTime.Now.ToString()}', '{id.Rows[0][0]}')";

dbConn.Open();

command = new SqlCommand(querytString, dbConn);

command.ExecuteNonQuery();

MessageBox.Show("Произведение сохранено!");

NOTES = NewNotes.Split(new char[] { ' ' });

DURATIONS = NewDurations.Split(new char[] { ' ' });

viewModel.SongNotes = NOTES;

CurrentNote = 0;

viewModel.CurrentNoteNumber = 0;

dbConn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

finally

{

dbConn.Close();

}

}

CurrentNote = 0;

viewModel.CurrentNoteNumber = 0;

}

public void SaveResults(string userID, string lessonID, string success)

{

using (SqlConnection dbConn = new SqlConnection())

{

try

{

dbConn.ConnectionString = ConfigurationManager.ConnectionStrings["YourPianoConnection"].ConnectionString;

string querytString = $"insert ProgressInfo(userID, lessonID, date, successRate) values ('{userID}', '{lessonID}', '{DateTime.Now.ToString()}', '{success}')";

dbConn.Open();

SqlCommand command = new SqlCommand(querytString, dbConn);

command.ExecuteNonQuery();

dbConn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

finally

{

dbConn.Close();

}

}

}

public async void PlaySingleReverseNote()

{

if (!IsPlaying)

{

try

{

IsPlaying = true;

FreePlayViewModel viewModel = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

if (CurrentNote == 0)

{

CurrentNote = NOTES.Length - 1;

viewModel.CurrentNoteNumber = CurrentNote;

}

else

{

CurrentNote -= 1;

viewModel.CurrentNoteNumber = CurrentNote;

}

viewModel.CurrentNoteNumber = CurrentNote;

SetPressedButtonColor();

MidiPlayer.Play(new NoteOn(0, 0, NOTES[CurrentNote], volume));

await Task.Factory.StartNew(() =>

{

Thread.Sleep(100);

});

SetOriginalButtonColor();

MidiPlayer.Play(new NoteOff(0, 0, NOTES[CurrentNote], volume));

IsPlaying = false;

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

}

public async void PlaySingleNote()

{

try

{

if (!IsPlaying)

{

IsPlaying = true;

FreePlayViewModel viewModel = (FreePlayViewModel)PageController.GetViewModel().CurrentPage.DataContext;

if (CurrentNote == NOTES.Length - 1)

{

CurrentNote = 0;

viewModel.CurrentNoteNumber = CurrentNote;

}

else

{

CurrentNote += 1;

viewModel.CurrentNoteNumber = CurrentNote;

}

viewModel.CurrentNoteNumber = CurrentNote;

SetPressedButtonColor();

MidiPlayer.Play(new NoteOn(0, 0, NOTES[CurrentNote], volume));

await Task.Factory.StartNew(() =>

{

Thread.Sleep(150);

});

SetOriginalButtonColor();

MidiPlayer.Play(new NoteOff(0, 0, NOTES[CurrentNote], volume));

IsPlaying = false;

}

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

public void ChangeInstrument(GeneralMidiInstruments targetInstrument)

{

PageController.GetPianoRollModel().ChangeInstrument(targetInstrument);

}

public void ChangeVolume(byte vol)

{

volume = vol;

PageController.GetPianoRollModel().ChangeVolume(vol);

}

public void ChangeLessonVolume(byte vol)

{

volume = vol;

PageController.GetLessonPianoRollModel().ChangeVolume(vol);

}

}

}