МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 05 03 Информационные системы и технологии Направление специальности 1-40 01 05 03 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное программирование»

Тема Программное средство «Помощник председателя товарищества собственников жилья»

Исполнитель

студентка 2 курса группы 1 Кашперко Василиса Сергеевна

(Ф.И.О.)

Руководитель работы ассистент Панченко О.Л.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра программной инженерии

Утверждаю

Заведующий кафедрой

Н.В. Пацей

подпись инициалы и фамилия

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Специальность: 1-40 01 05 03 Информационные системы и  тех­нологии  Студент: Кашперко В.С. | Группа: 1 |
| **Тема:** Программное средство «Помощник председателя товарищества собствен­ников жилья» | |

**1. Срок сдачи студентом законченной работы**: "20 мая 2022 г."

**2. Исходные данные к проекту:**

**2.1**. Функционально ПС поддерживает:

* Функции председателя ТСЖ:
  + Выполнять регистрацию и авторизацию;
  + Поддерживать работу c базой данных;
  + Добавлять и изменять информацию о жилых домах и собственниках квартир;
  + Просматривать информацию о жилых домах и собственниках квартир;
  + Выполнять поисковые запросы;
  + Добавлять контакты работников дома и организаций в записную книжку;
  + Выполнять регистрацию бухгалтера для определенного дома.
* Функции бухгалтера ТСЖ:
  + Выполнять авторизацию;
  + Просматривать информацию о собственниках квартир;
  + Добавлять и обновлять сведения о текущих задолженностях собственников.

**2.2.** При выполнении курсового проекта необходимо использовать принципы проектирования ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение. Отображение, бизнес логика должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, взаимодействия разработать на основе UML. Язык разработки проекта – C#. Управление программой должно быть интуитивно понятным и удобным. При разработке использовать несколько наиболее подходящих шаблонов проектирования ПО.

**3. Содержание расчетно-пояснительной записки**

(перечень вопросов, подлежащих разработке)

* Введение
* Постановка задачи и обзор литературы (алгоритмы решения, обзор прототипов, актуальность задачи)
* Проектирование архитектуры проекта (структура модулей, классов).
* Разработка функциональной модели и модели данных ПС (выполняемые функции)
* Тестирование
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления выполненного курсового проекта:**

* + Теоретическая часть курсового проекта должна быть представлена в формате docx. Оформление записки должно быть согласно выданным правилам.
  + Листинги программы представляются частично в приложении.
  + Пояснительную записку, листинги, проект (инсталляцию проекта) необходимо загрузить на диск, указанный преподавателем.

#### Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 19.02.2022 |  |
| 2 | Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования | 12.03.2022 |  |
| 3 | Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных) | 26.03.2022 |  |
| 4 | Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса | 02.04.2022 |  |
| 5 | Кодирование программного средства | 23.04.2022 |  |
| 6 | Тестирования и отладка программного средства | 30.04.2022 |  |
| 7 | Оформление пояснительной записки | 07.05.2022 |  |
| 8 | Сдача проекта | 20.05.2022 |  |

**5. Дата выдачи задания** 14.02.2022 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *О.Л. Панченко*

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *В.С. Кашперко*

(дата и подпись студента)

**Содержание**

[**ВВЕДЕНИЕ 6**](#_Toc104885362)

[1 Аналитический обзор прототипов и литературных источников 8](#_Toc104885363)

[2 Анализ требований к программному средству и разработка функциональных требований 11](#_Toc104885364)

[3 Проектирование программного средства 13](#_Toc104885365)

[3.1 Проектирование архитектуры приложения 13](#_Toc104885366)

[3.2 Проектирование базы данных 15](#_Toc104885367)

[3.3 Проектирование доступа к базе данных 17](#_Toc104885368)

[3.4 Проектирование логики сценариев использования 18](#_Toc104885369)

[3.5 Проектирование доступа к сервисам приложения 19](#_Toc104885370)

[4 Реализация программного средства 20](#_Toc104885371)

[4.1 Реализация сущностей 20](#_Toc104885372)

[4.2 Реализация архитектуры MVVM 21](#_Toc104885373)

[4.3 Реализация доступа к сервисам приложения 22](#_Toc104885374)

[4.4 Реализация представления 23](#_Toc104885375)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 24](#_Toc104885376)

[6 Руководство по использованию 28](#_Toc104885377)

[ЗАКЛЮЧЕНИЕ 38](#_Toc104885378)

[СПИСОК ЛИТЕРАТУРЫ 39](#_Toc104885379)

## **ВВЕДЕНИЕ**

Информационные системы и технологии все увереннее становятся неотъемлемой частью всех сфер жизнедеятельности современного человека. Все потому, что именно они дают возможность современному человеку жить в комфорте и получать мгновенный доступ к нужной информации, не теряясь в «стопках бумаг», блокнотах и других бумажных носителях информации. Также информационные системы позволяют структурировать данные, в кратчайшие сроки отобрать искомую информацию наиболее удобными способами, избежать ошибок при заполнении данных, обеспечивая целостность, и предотвратить потерю информации в бытовых ситуациях.

В потребности организации подобных систем нуждаются не только государственные структуры, большие влиятельные компании и обширные сети, но и юридические, физические и должностные лица, то есть обычные люди со своей спецификой повседневных и профессиональных задач.

Не исключением является и человек, занимающий должность председателя товарищества собственников жилья (далее - председатель ТСЖ). Председатель ТСЖ — это выборное должностное лицо, которое обеспечивает выполнение решений правления ТСЖ. Главная задача председателя — защита интересов собственников квартир в любых инстанциях. Между председателем и товариществом заключается трудовой договор, где подробно описываются закрепленные за ним обязанности, например:

* знать действующее законодательство страны (нормативно-правовые акты, относящиеся к деятельности сообщества, устав ТСЖ и должностную инструкцию);
* контролировать своевременную оплату жильцами коммунальных услуг и членских взносов;
* реализовывать решения, принятые на общем собрании;
* контролировать поддержание порядка в жилом доме;
* управлять персоналом по хозяйственной работе;
* работать с документацией и быть осведомленным о всех собственниках квартир;
* предоставлять отчет о проделанной работе, тратах на поддержание общедомового имущества.

Целью курсового проектирования является разработка программного средства, удовлетворяющего профессиональные потребности председателя ТСЖ. Данное программное средство позволит решать необходимый спектр задач, связанный с добавлением, просмотром и поиском необходимой информации о собственниках квартир, добавлением и просмотром важных контактов служб, организаций и работников дома. Также программа будет предоставлять разрешение на вход для зарегистрированных председателем бухгалтеров, предоставляя информацию об определенном доме и собственниках квартир данного дома, что позволит избежать телефонных разговоров и ошибок при введении обновленной информации, распределяя обязанности председателя и бухгалтера.

Управление программой придерживается интуитивно понятного и удобного интерфейса.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом.

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

Хранение данных осуществляется в Microsoft SQL Server.

Для успешной реализации курсового проекта необходимо:

* провести обзор и анализ соответствующей литературы;
* ознакомиться с прототипами программных средств по выбранной теме;
* определить функциональные требования программного средства;
* определить и продумать структуру базы данных;
* спроектировать архитектуру проекта;
* реализовать программное средство;
* произвести тестирование программного средства;
* написать руководство пользователя.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

1 Аналитический обзор прототипов и литературных источников

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

На сегодняшний день аналогичных и доступных в сети программных средств под операционные системы, подходящие для компьютеров или ноутбуков, существует не большое количество.

Мною были рассмотрены несколько программных средств:

«Председатель ТСЖ» – программа предназначена для использования в товариществах собственников жилья (ТСЖ), которые ведут как расчет квартплаты и прочих коммунальных услуг, так и паспортный учет. Программа написана на встроенном языке программирования 1С.

Интерфейс «Председатель ТСЖ» представлен на рисунке 1.1.

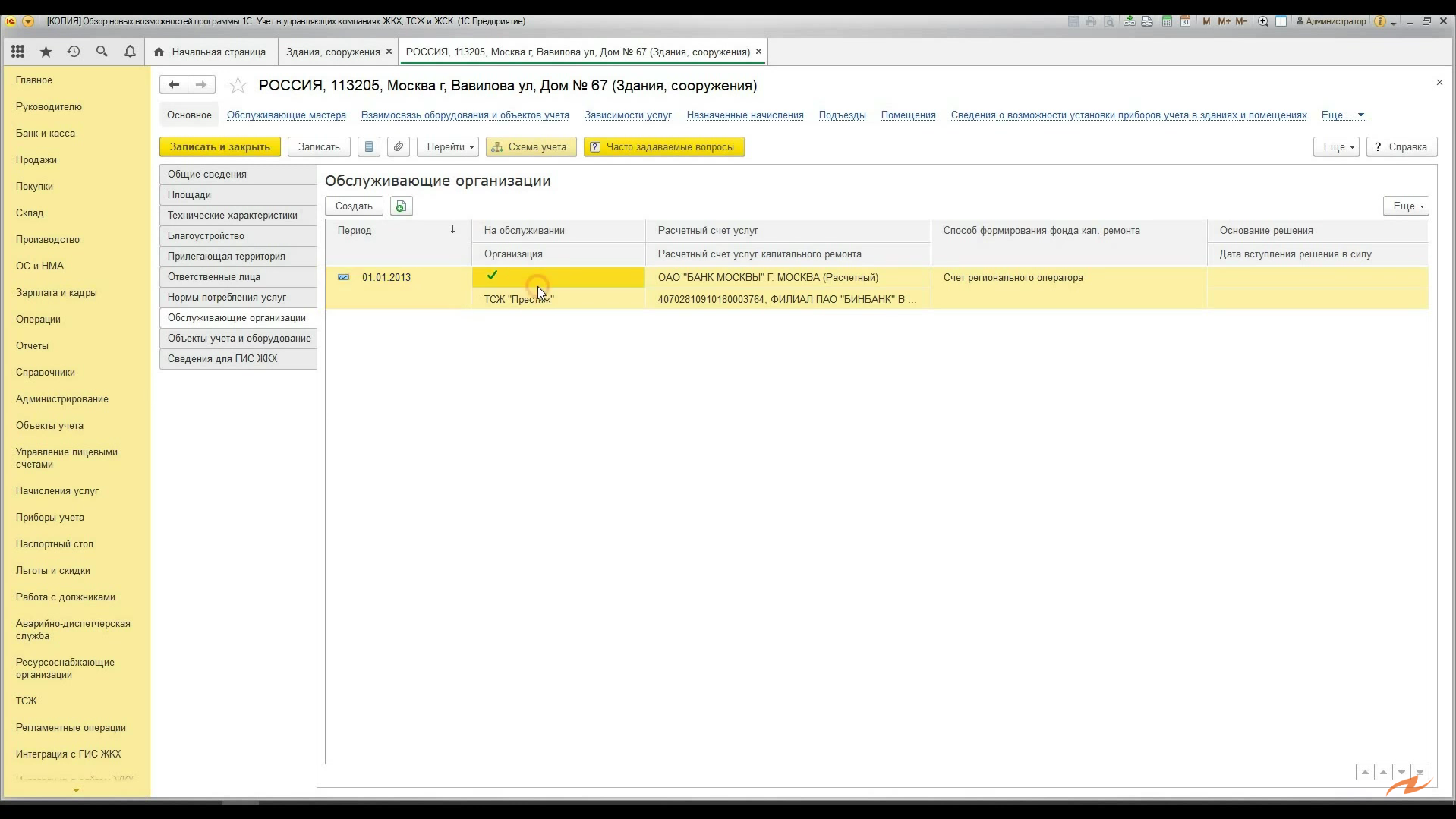


Рисунок 1.1. Интерфейс программного средства «Председатель ТСЖ»

Проанализировав «Председатель ТСЖ», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* сложный интерфейс;
* стоимость.

Основные плюсы:

* надежность;
* широкое использование не только среди председателей ТСЖ, но и ЖКХ и ЖСК;
* возможность установить мобильное приложение и синхронизировать данные.

«Председатель 365» - мобильное приложение, разработано для облегчения работы председателя ТСЖ. Воспользоваться приложением не удалось, так как после авторизации, предлагают только вариант платной подписки. Поэтому, изучить приложение мне помогли скриншоты и видео из видео-хостингов.

Интерфейс одной из доступных страниц приложения «Председатель 365» представлен на рисунке 1.2.

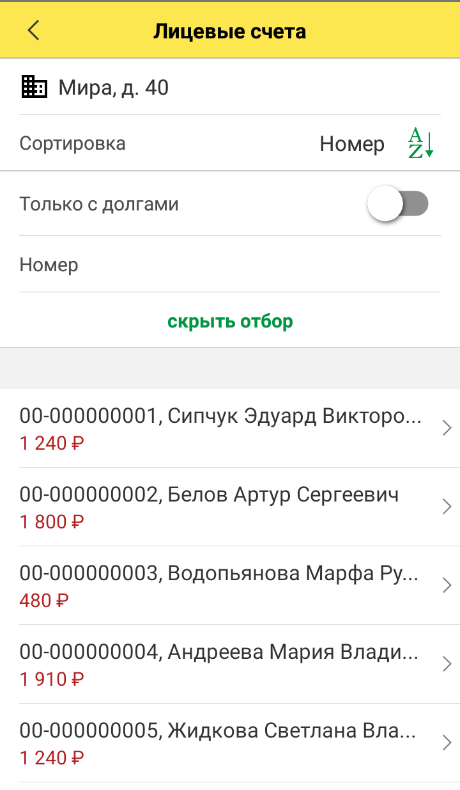


Рисунок 1.2 − Интерфейс «Председатель 365»

На рисунке 1.3 представлен скриншот окна со всеми адресами, введенными пользователем, приложения «Председатель 365»

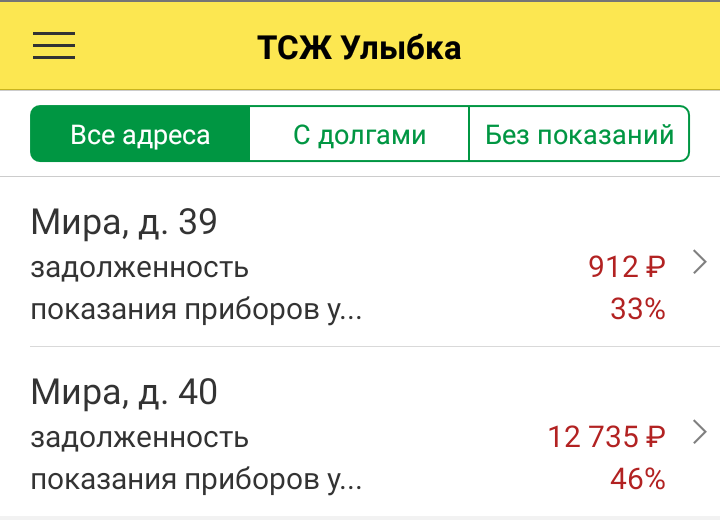


Рисунок 1.3 − Интерфейс «Председатель 365»

Проанализировав «Председатель 365», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* загруженный интерфейс;
* негативные отзывы покупателей подписки приложения;
* стоимость.

Основные плюсы:

* достаточно хороший UI;
* возможность загружать свои данные в облако.

«Домá» - мобильное приложение, разработано для облегчения работы председателя ТСЖ, помогающее своевременно получать заявки от жильцов, оплату жильцами членского взноса.

Интерфейс одной из доступных страниц приложения «Домá» представлен на рисунке 1.4.

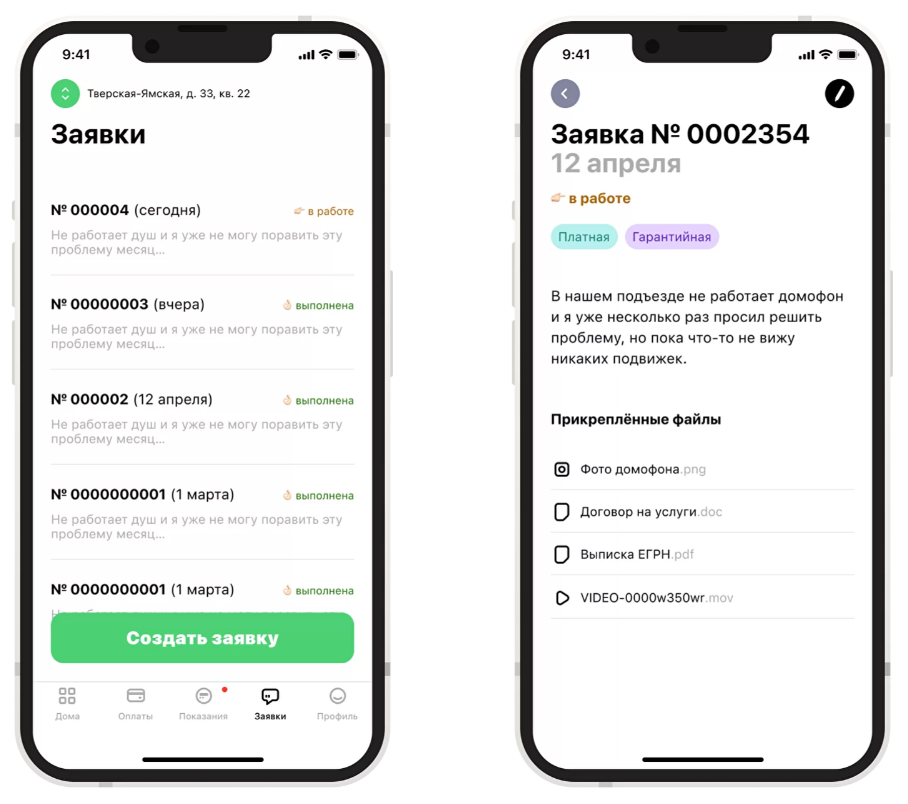


Рисунок 1.4 − Интерфейс «Домá»

Проанализировав «Председатель 365», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

* стоимость;
* в приложении нет функций, сокращающих бумажную работу председателя.

Основные плюсы:

* хороший UI;
* возможность загружать свои данные в облако;
* возможность получать заявки от жильцов;
* возможность контролировать оплату членских взносов.

2 Анализ требований к программному средству и разработка функциональных требований

Анализ требований — это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Цель анализа требований в проектах — получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски. На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Определение и описание требований — шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы.

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству. К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программи-рования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей приложения, а именно председателя ТСЖ и бухгалтера, закрепленного за председателем. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы. Пользователь данного программного решения должен иметь возможности, соответствующие его роли.

Возможности председателя ТСЖ:

* + выполнять регистрацию и авторизацию;
  + поддерживать работу c базой данных;
  + добавлять и изменять информацию о жилых домах и собственниках квартир;
  + просматривать информацию о жилых домах и собственниках квартир;
  + выполнять поисковые запросы;
  + добавлять контакты работников дома и организаций в записную книжку;
  + выполнять регистрацию бухгалтера для определенного дома.

Возможности бухгалтера ТСЖ:

* + выполнять авторизацию;
  + просматривать информацию о собственниках квартир;
  + добавлять и обновлять сведения о текущих задолженностях собственников.

После проведения анализа были выявлены следующие функциональные требования:

* архитектура приложения должна соответствовать шаблонам проектирования, таким как: MVVM, Command;
* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем;
* приложение должно предоставлять возможность поиска жильцов и квартир с соответствующими данными по следующим критериям, как: фамилия, номер квартиры.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

**3 Проектирование программного средства**

Проектирование программного средства — процесс создания проекта программного обеспечения. Целью проектирования является определение внутренних свойств системы и детализации её внешних свойств на основе исходных условий задачи. Исходные условия задачи уже были сформулированы во втором разделе данной пояснительной записки. Этап проектирования подразумевает их анализ.

## **3.1 Проектирование архитектуры приложения**

Архитектура программного обеспечения — совокупность важнейших решений об организации программной системы.

Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
* соединение выбранных элементов структуры и поведения во всё более крупные системы;
* архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). В разрабатываемом приложении используется архитектурный шаблон Model-View-ViewModel (MVVM).

Шаблон MVVM имеет три основных слоя: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

На рисунке 3.1 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.1 – Структура шаблона MVVM

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Так как пользовательский интерфейс и качество его реализации играет далеко не последнее место в конечном результате, разработка эффективного интерфейса, приятного и удобного для конечного пользователя, является важной задачей. Поэтому для хорошего проектирования View необходимо понять, как пользователь будет взаимодействовать с приложением.

Пользователь (председатель ТСЖ) после входа в приложение пользователю откроется окно приложения с 4 вкладками:

* Мои дома;
* Контакты;
* Справка;
* Выйти.

На странице «Мои дома» будет отображаться сетка с добавленными председателем домами, кнопки, предоставляющие возможность добавить и удалить бухгалтера. На этой странице пользователь сможет просмотреть добавленные дома, увидеть имя, закрепленного за ним бухгалтера, и, нажав по элементу сетки с домами, перейти на следующую соответствующую страницу. На последующих страницах, таких как: «Добавить дом», «Добавить жильца», «Изменить информацию о жильце» и возможностях кнопок реализующих удаление домов и жильцов, осуществляется вся важнейшая бизнес-логика, которая касается интересов работы пользователя.

В разделе «Контакты» отображается таблица с добавленными пользователем контактами. При нажатии на кнопку «Добавить» открывается страница «Добавить контакт». При нажатии на кнопку «Изменить» открывается страница «Изменить контакт».

На странице «Справка» можно ознакомиться с актуальными сведениями и цитатами из кодекса РБ о работе председателя ТСЖ.

В разделе «Выйти», нажав по нему левой кнопкой мыши дважды, пользователь сможет выйти из аккаунта. Спустя некоторое время ему предоставится возможность войти в аккаунт еще раз.

При авторизации пользователя с ролью «Бухгалтер» исчезают все пункты меня, кроме «Мои дома».

Для бухгалтера появляется доступ к страницам «Подробнее о доме», «Изменить текущую задолженность».

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Она содержит Модель, преобразованную к Представлению, а также команды, которыми может пользоваться Представление, чтобы влиять на Модель.

Model или Модель представляет собой логику работы с данными и описание фундаментальных данных, необходимых для работы приложения.

Для разработки приложения в качестве сервиса для хранения данных была выбрана СУБД Microsoft SQL Server. На диаграмме размещение на рисунке 3.3 показано графическое представление инфраструктуры приложения.

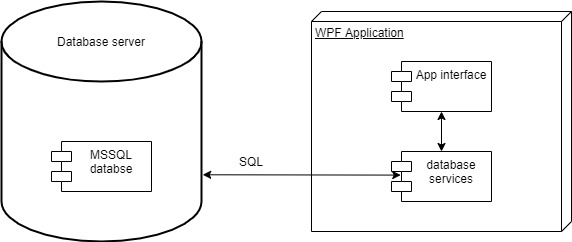


Рисунок 3.3 – Диаграмма размещения

## **3.2 Проектирование базы данных**

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.

Основные задачи проектирования базы данных:

* обеспечение хранения в БД всей необходимой информации;
* обеспечение возможности получения данных по всем необходимым запросам;
* сокращение избыточности и дублирования данных;
* обеспечение целостности базы данных.

Проектирование базы данных проводится в два этапа: концептуальное и логическое проектирование.

Концептуальное проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. В результате этого этапа создаётся ER-модель. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных.

Основными понятиями ER-модели являются: сущность, связь и атрибут

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация обычно является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь).

Атрибут сущности − это любая детaль, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

В рамках этого этапа была создана ER-модель, которая включает 8 сущностей:

* пользователь;
* дом;
* адрес;
* подъезд;
* квартира;
* жилец-собственник;
* номер телефона;
* контакт.

Также в ER-модели были определены необходимые связи. Например, между сущностями «пользователь» и «дом» была установлена связь один-ко-многим. Для каждой сущности были выделены атрибуты.

Например, для дома в качестве атрибутов были выделены такие характеристики, как «идентификатор дома», «идентификатор пользователя», «название дома», «количество квартир», «количество подъездов» и «идентификатор адреса».

Логическое проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных.

Для реляционной модели данных логическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Всего в базе данных содержится 8 таблиц.

В таблице Users хранятся все пользователи, зарегистрированные в приложении.

Таблица Houses содержит информацию о добавленных пользователем домах.

Таблица Addresses содержит информацию о адресах добавленных домов.

Таблица Porches хранит сгенерированные подъезды, чтобы упросить навигацию по квартирам в доме.

Таблица Flats содержит квартиры домов. Таблица Owners хранит информацию о собственниках квартир.

Таблица PhoneNumbers хранит телефонные номера как собственников, так и людей, добавленных в контакты.

Таблица Contacts содержит информацию о добавленных контактах.

Логическая модель базы данных представлена на рисунке 3.4

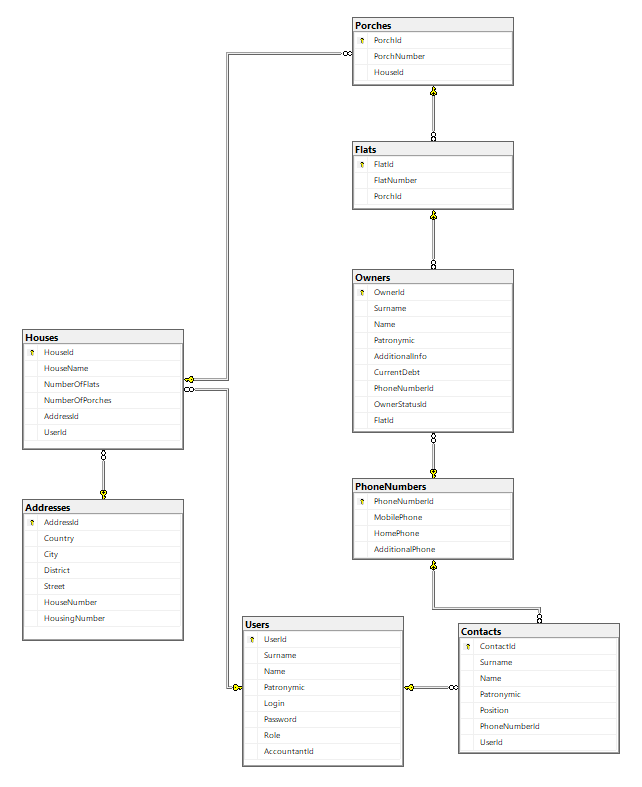


Рисунок 3.4 - Логическая модель базы данных

## **3.3 Проектирование доступа к базе данных**

Для доступа к базе данных используется Entity Framework 6. Этот подход предоставляет ряд существенных преимуществ: нам не нужно беспокоиться о коде доступа к данным и не нужно знать деталей работы СУБД SQL Server и синтаксиса языка запросов T-SQL, вместо этого мы работаем с таблицами базы данных как с классами C#, с полями этих таблиц - как со свойствами классов, а синтаксис SQL-запросов заменен на  [LINQ](https://professorweb.ru/my/LINQ/base/level1/info_linq.php).

Entity Framework берет на себя обязанности по преобразованию кода C# в SQL-инструкции.

Существуют 3 подхода по проектированию базы данных:

* Database-First;
* Model-First;
* Code-First.

В своей работе я использовала подход Code-First, при данном подходе модель EDMX не используется, и мы вручную настраиваем классы C# объектной модели (данный подход поддерживает как генерацию сущностных классов из существующей базы данных, так и создание базы данных из созданной вручную модели объектов C#).

## **3.4 Проектирование логики сценариев использования**

При проектировании приложения были рассмотрены различные сценарии использования. На рисунке 3.5 представлена диаграмма последовательности добавления нового жильца в квартиру.

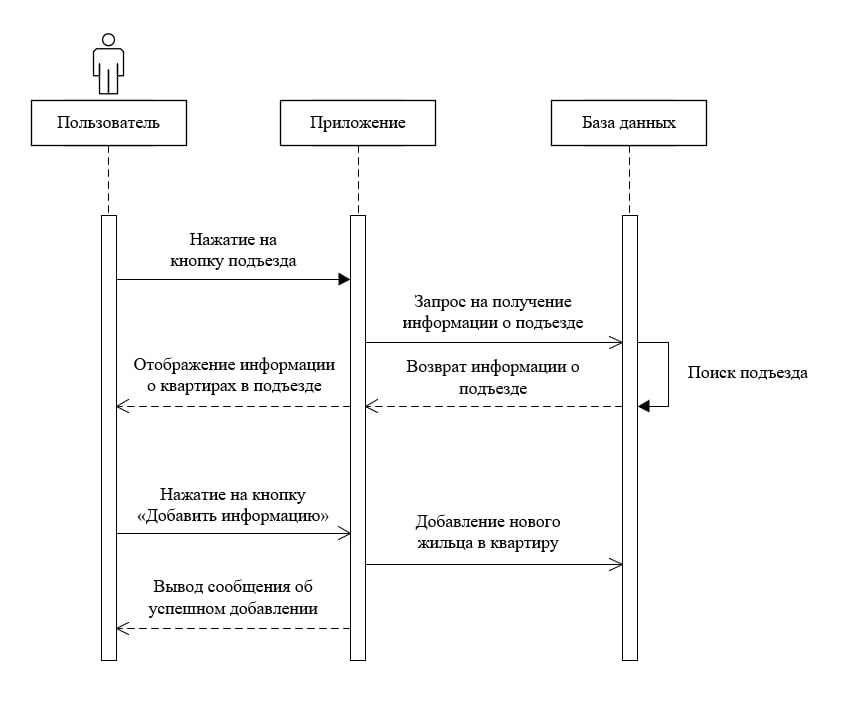


Рисунок 3.5 − Диаграмма последовательности добавления нового жильца в квартиру

Из диаграммы видно, что сначала пользователю необходимо нажать на кнопку «Подъезд». Затем приложение отправит запрос в базу данных на получение информации об этом подъезде. Дальше, когда база данных вернет информацию о выбранном подъезде, эта информация будет показана пользователю на этой же странице «Информация о доме». На этой странице присутствует кнопка добавления нового жильца. После указания нужной квартиры по нажатию этой кнопки открывается страница «Добавить жильца», и, после добавления, приложение отправит запрос на добавление этого жильца в соответствующую таблицу, для соответствующей квартиры. После добавления, пользователю будет выведено сообщение об успешном добавлении.

## **3.5 Проектирование доступа к сервисам приложения**

Приложение использует принцип Inversion of Control. Inversion of Control (инверсия управления) – это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

Для доступа к сервисам навигации в приложении используется IoC-контейнер.

В ViewModel средства навигации будут встраиваться через Dependency Injection(внедрение зависимостей), а именно через паттерн внедрения зависимости через конструктор ([Constructor Injection](http://sergeyteplyakov.blogspot.com/2012/12/di-constructor-injection.html)).

Внедрение зависимостей (DI, Dependency Injection) – это механизм передачи классу его зависимостей. Суть паттерна [Constructor Injection](http://sergeyteplyakov.blogspot.com/2012/12/di-constructor-injection.html) сводится к тому, что все зависимости, требуемые некоторому классу передаются ему в качестве параметров конструктора, представленных в виде интерфейсов или абстрактных классов.

Для доступа к IoC-контейнерам и отдельным ViewModel используется паттерн Service Locator. Основная идея паттерна Service Locator заключается в том, чтобы иметь объект, который знает, как получить все сервисы, которые, возможно, потребуются. Главной особенностью Service Locator является то, что он не создаёт объекты, а знает, как получить тот или иной объект.

MD5 - это алгоритм, позволяющий получить своего рода "отпечаток" исходной строки. Он спроектирован таким образом, чтобы из получившегося отпечатка нельзя было восстановить исходную строку (т.е. зашифровать можно, а расшифровать нет). Алгоритм широко используется при авторизации.

Также у MD5, как у любой хэш-функции, существует такое понятие как коллизии — это получение одинаковых хэшей для разных исходных строк.

алгоритм, позволяющий находить коллизии на обычном компьютере с любым начальным вектором (A,B,C,D) при помощи метода, названного им «туннелирование».

4 Реализация программного средства

Следующим этапом разработки приложения является непосредственная реализация программного решения в соответствии с уже сформированными требованиями и шаблонами.

4.1 Реализация сущностей

В соответствии с требованиями в качестве хранилища данных программного средства должна быть база данных, поэтому первым шагом в реализации программы является выбор технологии, позволяющей это осуществить. Выбор остановился на ORM технологии Entity Framework. Она предоставляет три подхода по проектированию базы данных. В данном программном решении был использован подход Code-First. При данном подходе модель EDMX не используется. Создание базы данных происходит из созданной вручную модели объектов C#. Созданные модели объектов совпадают с сущностями, которые были сформированы раннее в разделе 3.2.

Диаграмма классов UML для сущностных классов представлена на рисунке 4.1.

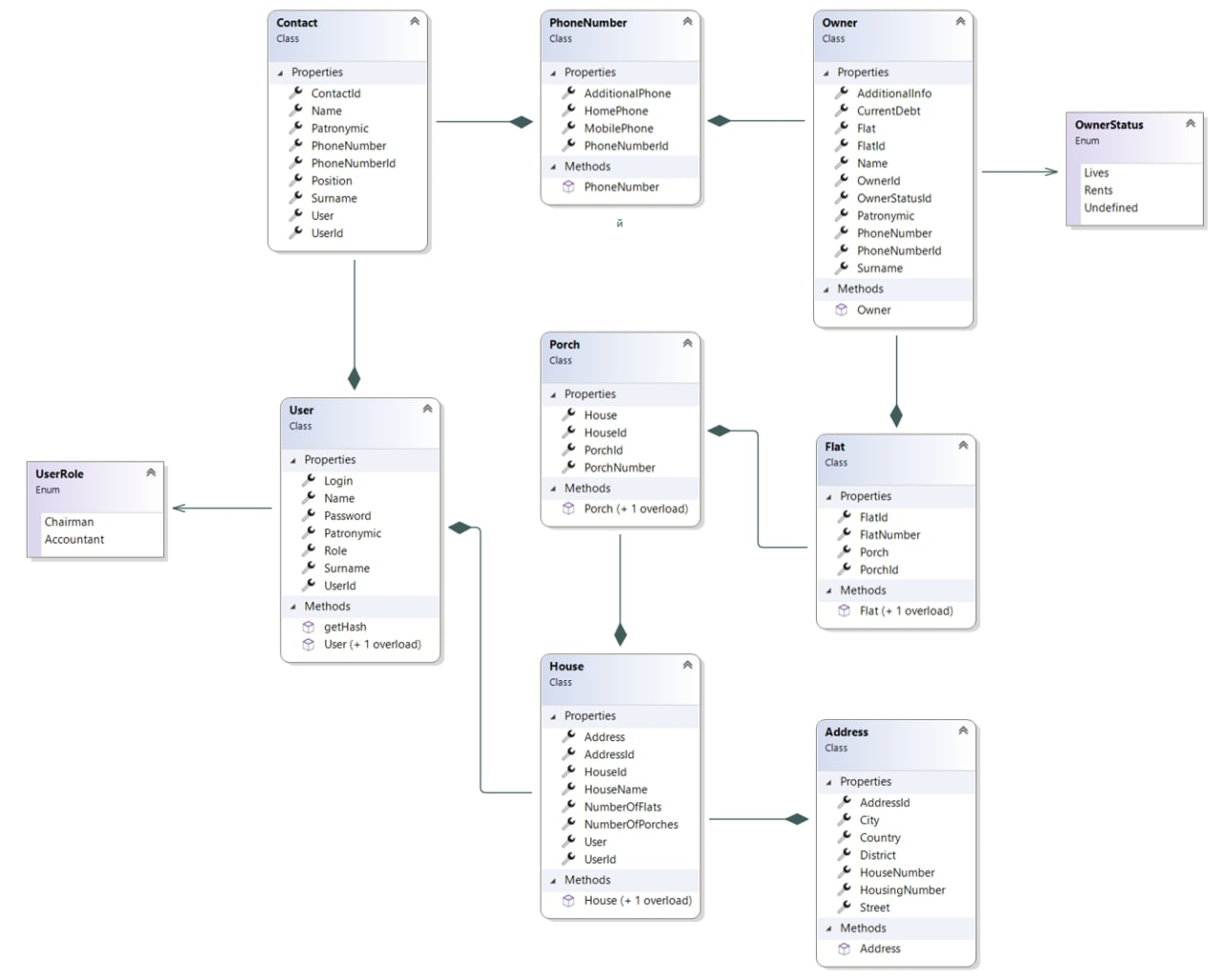


Рисунок 4.1 − Диаграмма классов

На диаграмме классов видно, как связаны между собой сущности. Сущность User связана с классами House, Contact отношением композиции т.е. при уничтожении объекта User в области памяти вместе с ним будут уничтожены и объекты House, Contact связанные с ним. И в этом плане объект User является главным, а объекты House, Contact– зависимыми. Также на диаграмме можно заметить, что остальные классы связаны таким же принципом.

Из диаграммы видно, что класс User и перечисление UserRole, а также Owner и OwnerStatus связаны между собой отношением ассоциации. Перечисление UserRole показывает роль текущего пользователя: 0 – председатель, 1 – бухгалтер. Перечисление OwnerStatus показывает статус собственника: 1 - Проживает, 2 – Сдает, 3 – Не выбрано.

4.2 Реализация архитектуры MVVM

Для ускорения создания и разработки приложений с архитектурой MVVM существует ряд различных библиотек. В качестве такой библиотеки в данном решении был выбран инструментарий MVVM Light Toolkit.

Для реализации паттерна MVVM файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули представлено на рисунке 4.3.

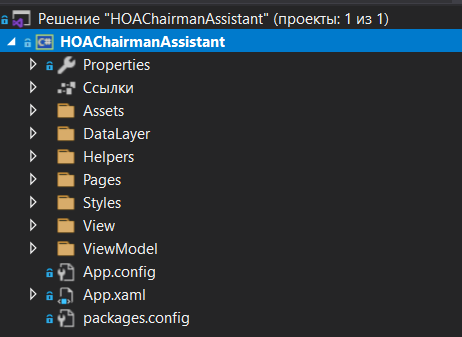


Рисунок 4.3 – Логические модули проекта

В папке Assets находятся картинки, использующиеся в приложении.

В папке DataLayer содержатся классы-сущности, которые используются для создания БД. Их описание представлено в разделе 4.1.

В папке Helpers находятся хранятся различные AttachedProperty, вспомогательные классы перечислений, конвертеры, пользовательские классы валидации, классы для передачи сообщений с помощью библиотеки MVVM Light Toolkit. Также в эту папку вынесены классы для работы с навигацией по страницам и определение для команды.

Навигация по страницам представлена классом FrameNavigationService который реализует интерфейс IFrameNavigationService и INotifyPropertyChanged. Интерфейс INotifyPropertyChanged позволяет уведомить систему об изменениях свойств модели. Интерфейс IFrameNavigationService обязует класс навигации иметь свойство Parameter и реализовать интерфейс INavigationService который находится в пространстве имен GalaSoft.MvvmLight.Views (библиотеки MVVM Light). Интерфейс INavigationService обязует иметь свойство CurrentPageKey которое отвечает за текущую страницу в окне. Метод GetBack – позволяет вернуться назад на 1 страницу. Метод NavigateTo открывает страницу, в окне в котором произошел вызов этого метода. Код класса навигации FrameNavigationService представлен в приложении Б.

В приложении используется паттерн Command который позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. В WPF команды представлены интерфейсом ICommand. В приложении он представлен в виде RelayCommand и собственной команды RelayCommandParametr код которой приведен в приложении В. Класс реализует два метода:

* CanExecute: определяет, может ли команда выполняться
* Execute: собственно, выполняет логику команды

А также реализует событие CanExecuteChanged вызывается при изменении условий, указывающий, может ли команда выполняться. Для этого используется событие CommandManager.RequerySuggested.

В папке Styles определены стили для представлений.

В папке ViewModels находятся все классы ViewModel для каждого представления. Также там содержится класс ViewModelLocator код которого предоставлен в приложении Г, а описание предоставлено в следующем подразделе.

В папке Pages хранятся все страницы программного средства. В папке Views хранятся все используемые представления.

**4.3 Реализация доступа к сервисам приложения**

В ViewModelLocator реализуется паттерн Service Locator, с помощью статического класса ServiceLocator, который использует в качестве провайдера IoC-контейнер SimpleIoC (библиотеки MVVM Light). В контейнере регистрируются 4 ViewModel: MainViewModel, AdminViewModel, CookViewModel, LoginWindowViewModel. А также регистрируется сервис навигации страниц FrameNavigationService которому передается Uri необходимых page и ключи по которым можно обратиться к ним. Сам класс FrameNavigationService описан в пункте 4.2.

Реализацию паттерна [Constructor Injection](http://sergeyteplyakov.blogspot.com/2012/12/di-constructor-injection.html) можно посмотреть на примере MainViewModel код класса приведён в приложении Д. В классе определенно поле IFrameNavigationService \_navigationService значение которому присваивается через конструктор.

**4.4 Реализация представления**

Для разработки графической части приложения была выбрана технология WPF.

Windows Presentation Foundation (WPF) — это библиотека для создания пользовательских интерфейсов для интеллектуальных клиентских приложений. В основе WPF лежит мощная инфраструктура, основанная на DirectX — API-интерфейсе графики с аппаратным ускорением, который обычно используется в современных компьютерных играх. Это означает, что применение развитых графических эффектов не приведёт к снижению производительности.

Одной из важных особенностей WPF является использование языка декларативной разметки интерфейса XAML, основанного на XML. Разработка с использованием XAML позволяет отделить графический интерфейс от логики приложения, а также создавать насыщенный интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#.

В конечном итоге в приложении было реализовано два окна:

* стартовое окно (регистрация и вход в систему);
* главное окно (содержит различные страницы);

А также четырнадцать страниц:

* страница для авторизации;
* стартовая для регистрации;
* главная страница со всеми домами пользователя;
* страница для добавления новых домов;
* страница для просмотра информации о конкретном доме;
* страница добавления пользователя;
* страница изменения информации о пользователе и его удалении;
* страница для отображения справки;
* страница для отображения добавленных контактов;
* страница для добавления новых контактов;
* страница для изменения и удаления контактов;
* страница для добавления бухгалтера;
* страница для просмотра всех домов закрепленного председателя (для бухгалтера);
* страница для изменения информации о текущей задолженности собственника (для бухгалтера).

В результате выполнения данного этапа было создано функционирующее программное средство.

5 Тестирование, проверка работоспособности и анализ полученных результатов

Прежде всего были проведены тесты авторизации: проверка на пустые и неверные данные. Валидация выполнена таким образом, что кнопка для входа или регистрации будет недоступна до тех пор, пока введённые данные не будут корректными. При таком способе валидации шанс ввести невалидные данные минимален. На рисунке 5.1 видно, что валидация на странице регистрации не допускает ввода некорректных полей. Запрещается вводить пробелы в поля имение, логина и пароля. В поле «Имя» можно ввести только буквы латинского и русского алфавитов длинною в 2 символа минимум и 30 символов максимум, и это поле является обязательным, так же как логин и пароль. Похожая валидация назначена и на другие поля. Логин может содержать только латинские буквы и цифры с специальными символами ‘.’ и ‘\_’. Длина поля от 5 до 40 символов. Пароль должен содержать только латинские буквы, минимум 1 букву и цифру, а также спец символы.

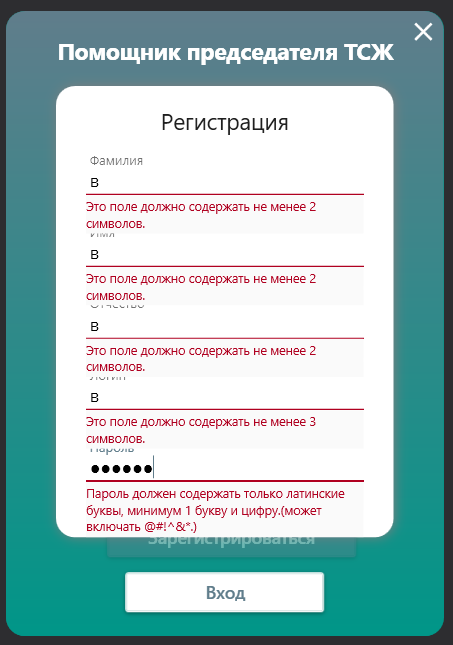
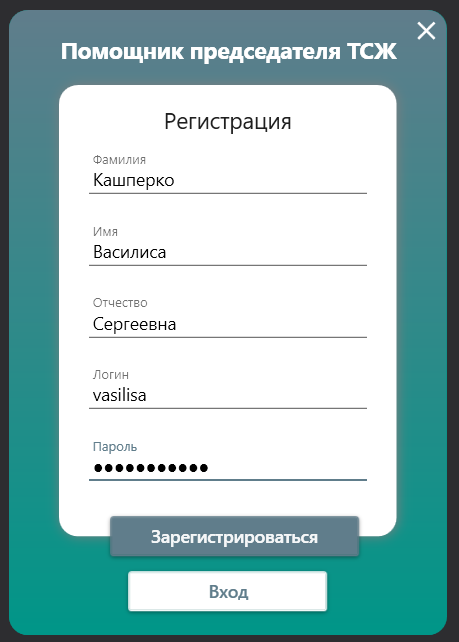
 

Рисунок 5.1 – Валидация регистрационной формы

На рисунке 5.2 представлена валидация формы входа.

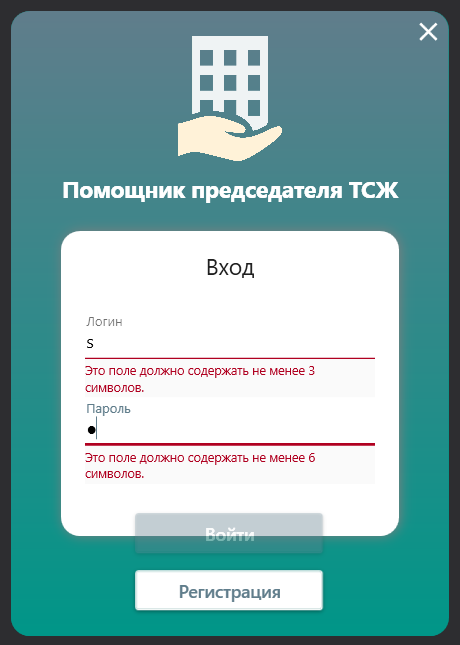


Рисунок 5.2 – Валидация формы входа

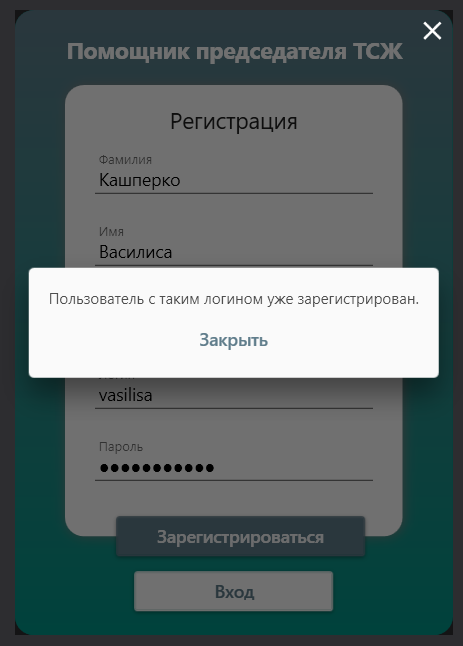


Рисунок 5.3 – Регистрация уже существующего пользователя

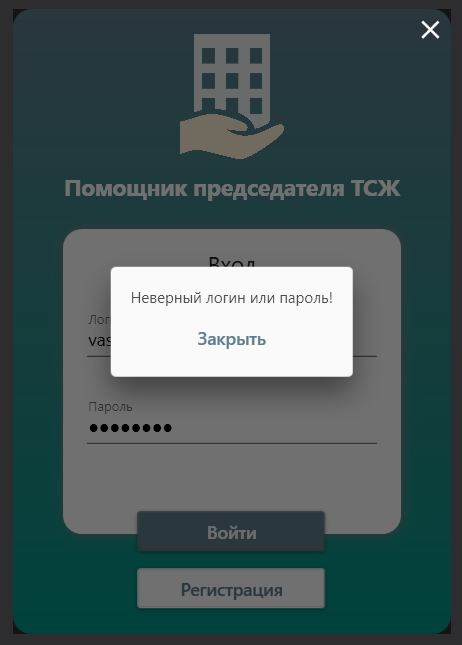


Рисунок 5.4 – Вход с неверным логином или паролем

Также были проведены тесты на работу валидации внутри приложения. Наличие валидации является обязательным в связи с тем, что в приложении постоянно ведётся работа с базой данных. Валидация организованна таким же образом, как и на форме авторизации, т.е. кнопки добавить и обновить будут недоступны до тех пор, пока не будут введены валидные данные. Все ошибки, возникшие при валидации, доступны и отображены удобным образом, что позволяет пользователю легко понять какие данные не валидны и быстро исправить их.

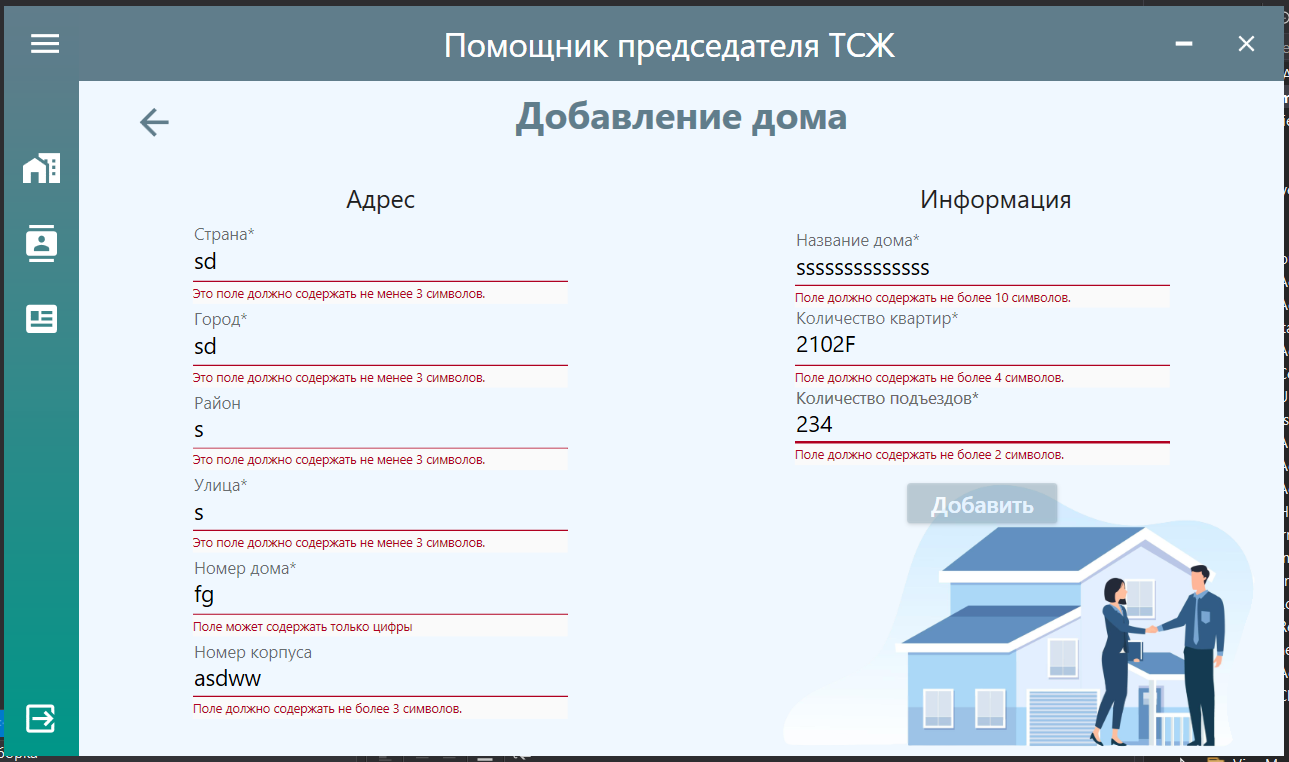


Рисунок 5.5 – Валидация при добавлении дома

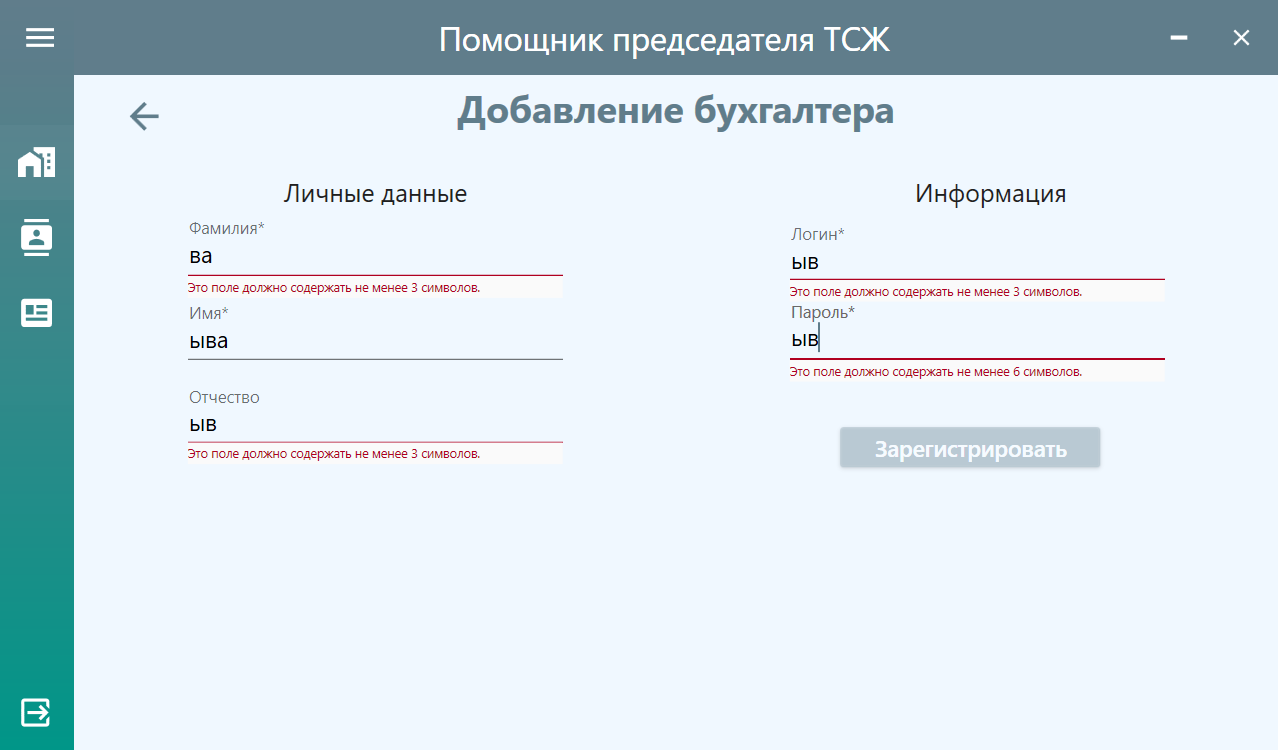


Рисунок 5.6 – Валидация при добавлении бухгалтера

Однотипная валидация предоставлена во всех полях для ввода данных.

На данном этапе были выполнены тесты на проверку работоспособности приложения, а именно на проверку валидации. Были проведены анализы результатов, которые показали, что валидация в приложении работает в соответствии с тем, как она задумывалась.

6 Руководство по использованию

После запуска приложения в первую очередь открывается окно входа и регистрации, на которой мы регистрируемся и авторизуемся.

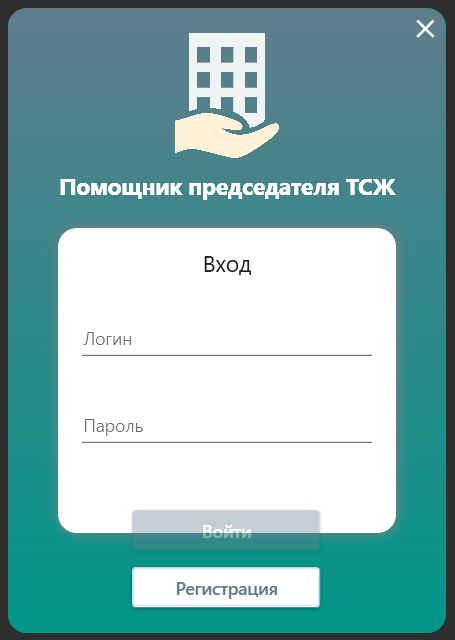


Рисунок 6.1 – Окно входа

После того, как мы вошли в аккаунт, открывается главная страница с уже добавленными домами или возможностью добавления нового дома, если пользователь только зарегистрировался.

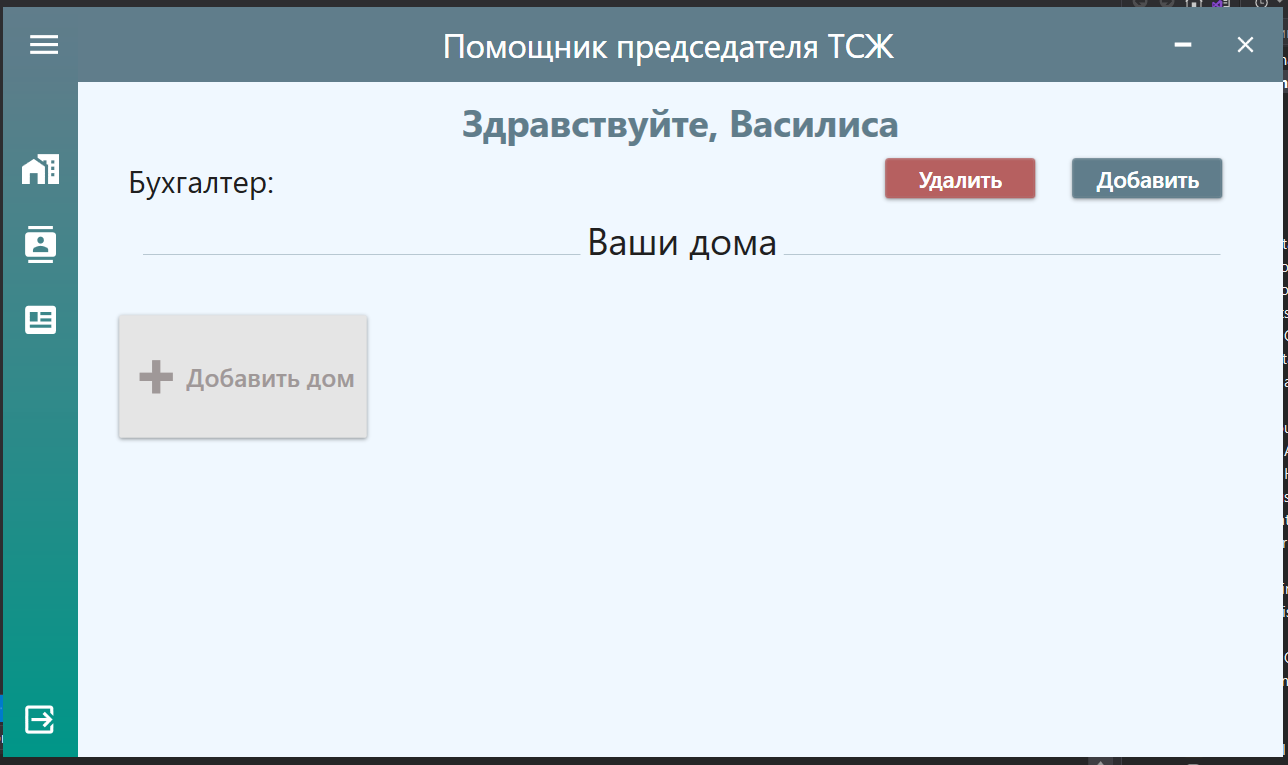


Рисунок 6.2 – Главная страница нового пользователя

Далее пользователь может добавить дом.

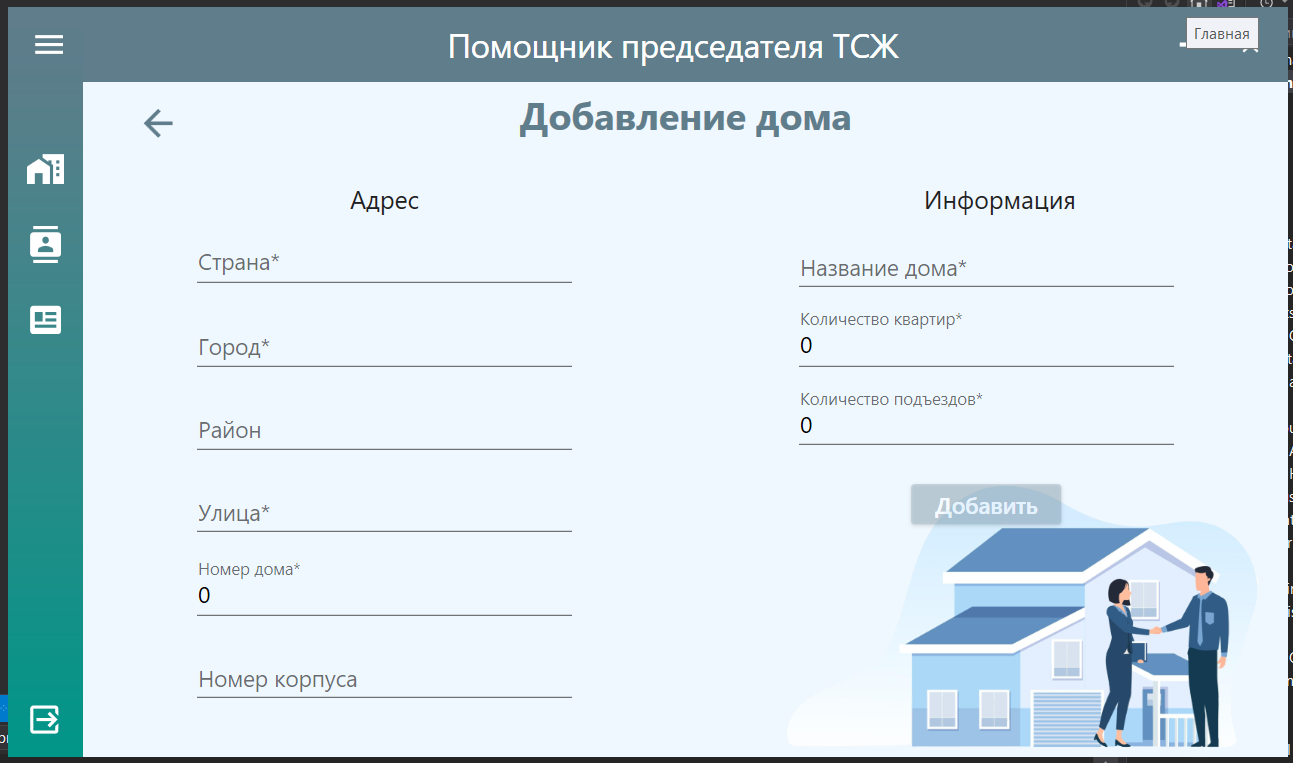


Рисунок 6.3 – Добавление дома

После добавление дома у пользователя появится в поле зрения ранее добавленный дом.

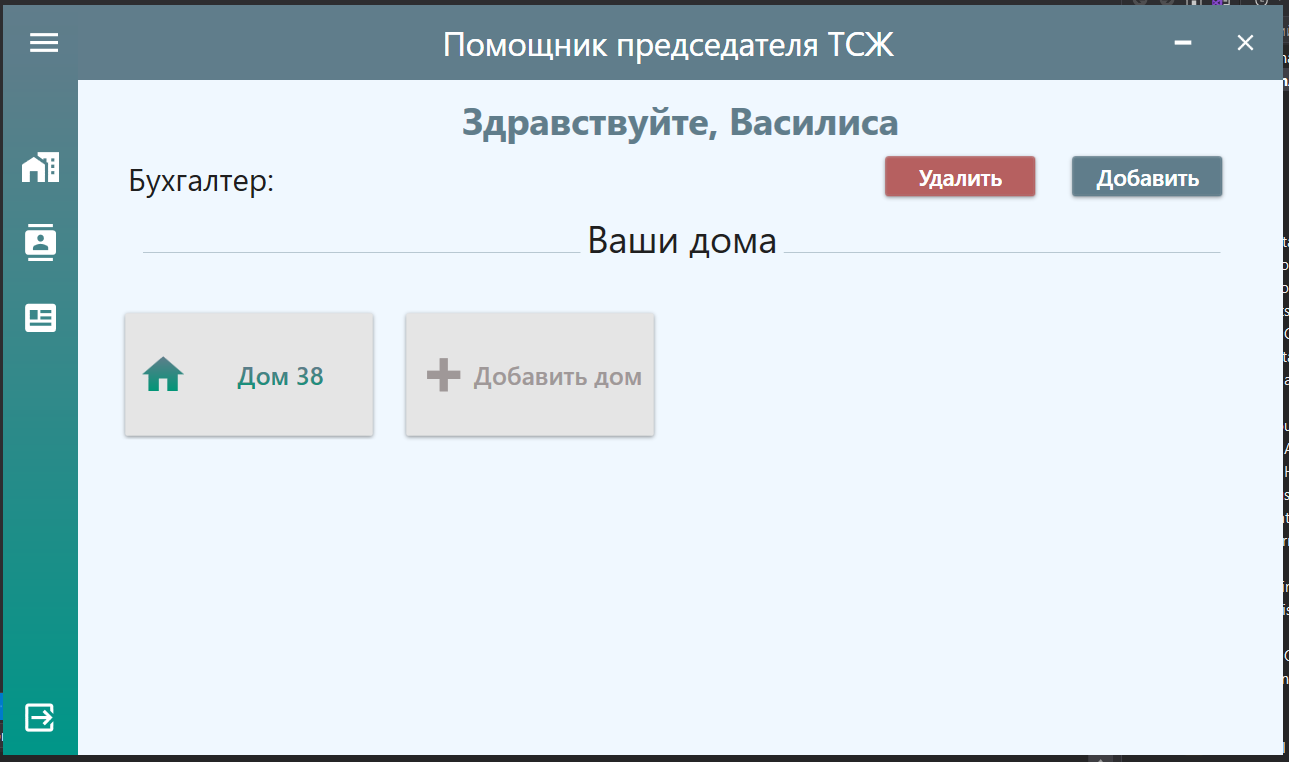


Рисунок 6.4 – Страница домов после добавление дома

Затем, можно добавить бухгалтера по нажатию кнопки «Добавить» напротив строки с надписью «Бухгалтер».

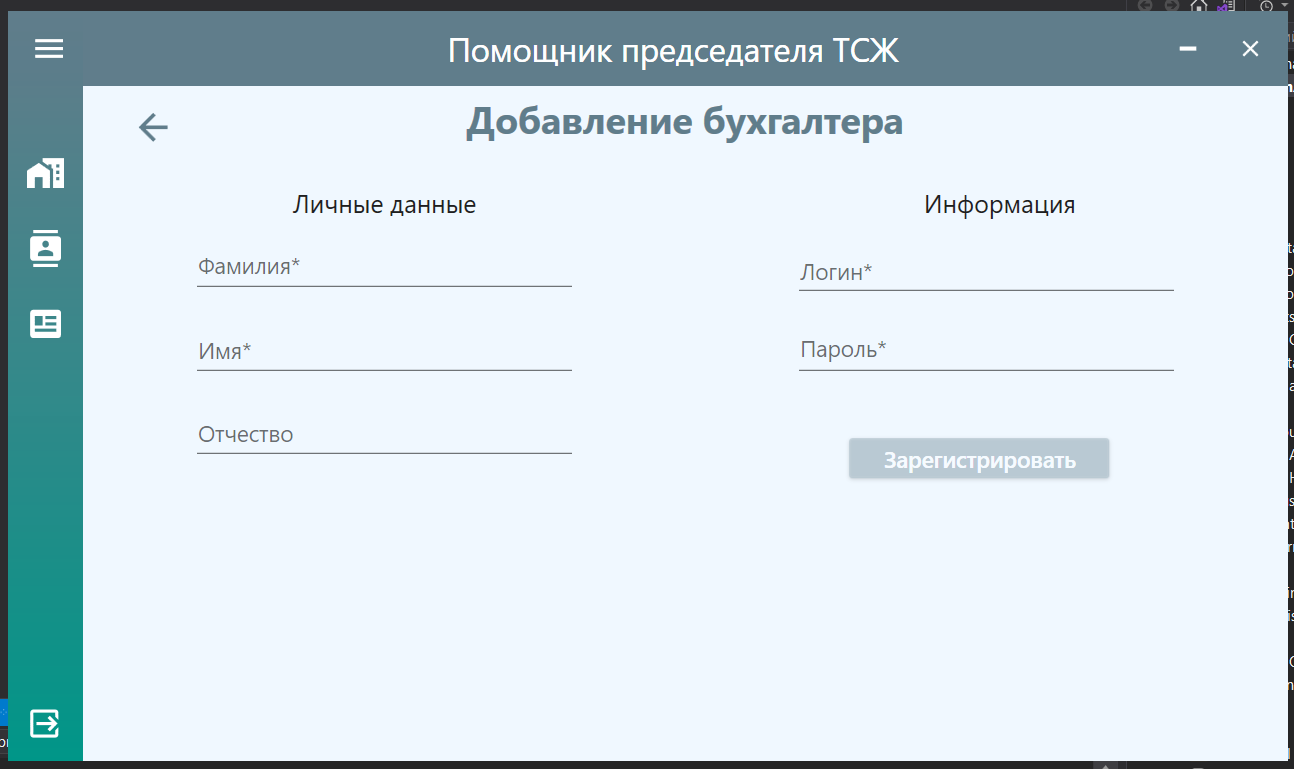


Рисунок 6.5 – Страница добавления бухгалтера

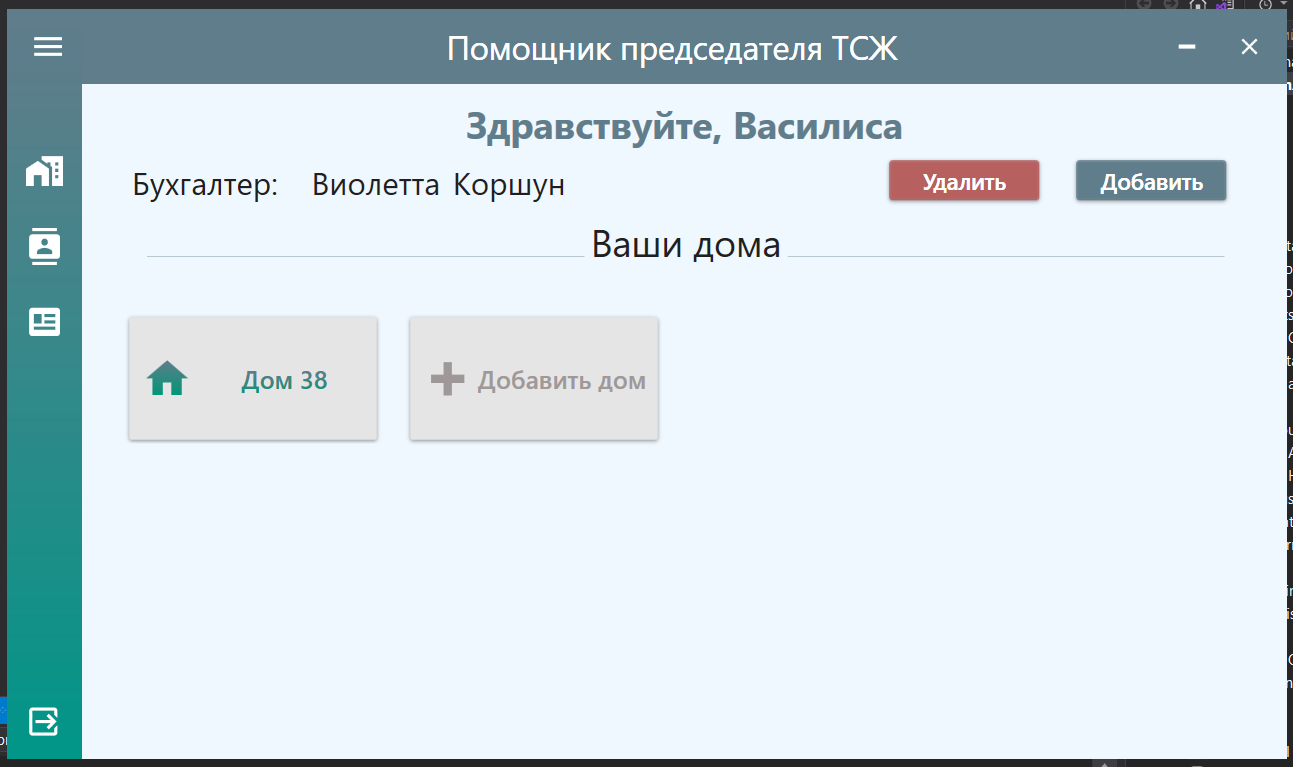


Рисунок 6.5 – Страница домов после добавления бухгалтера

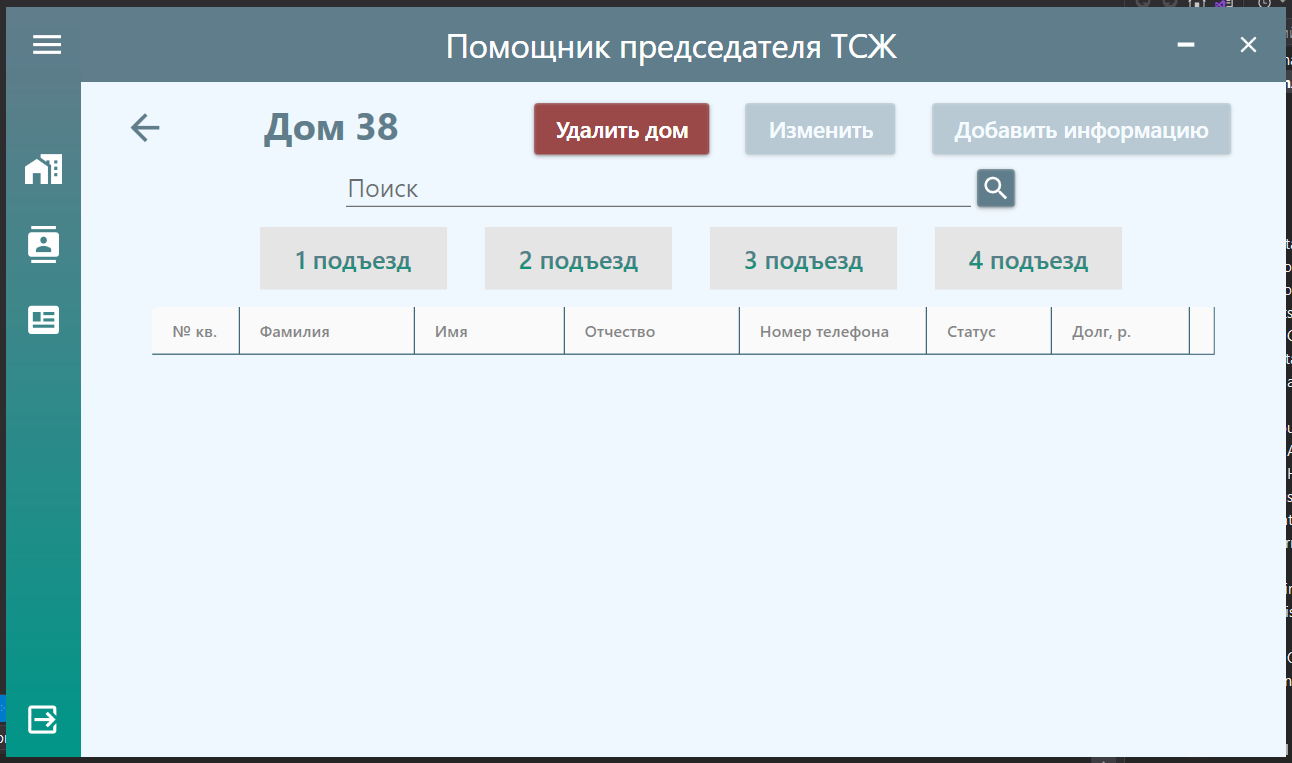


Рисунок 6.6 – Страница просмотра квартир дома

По нажатию на кнопку «Подъезд», автоматически генерируется запрос в базу данных для поиска подъезда в определенном доме.

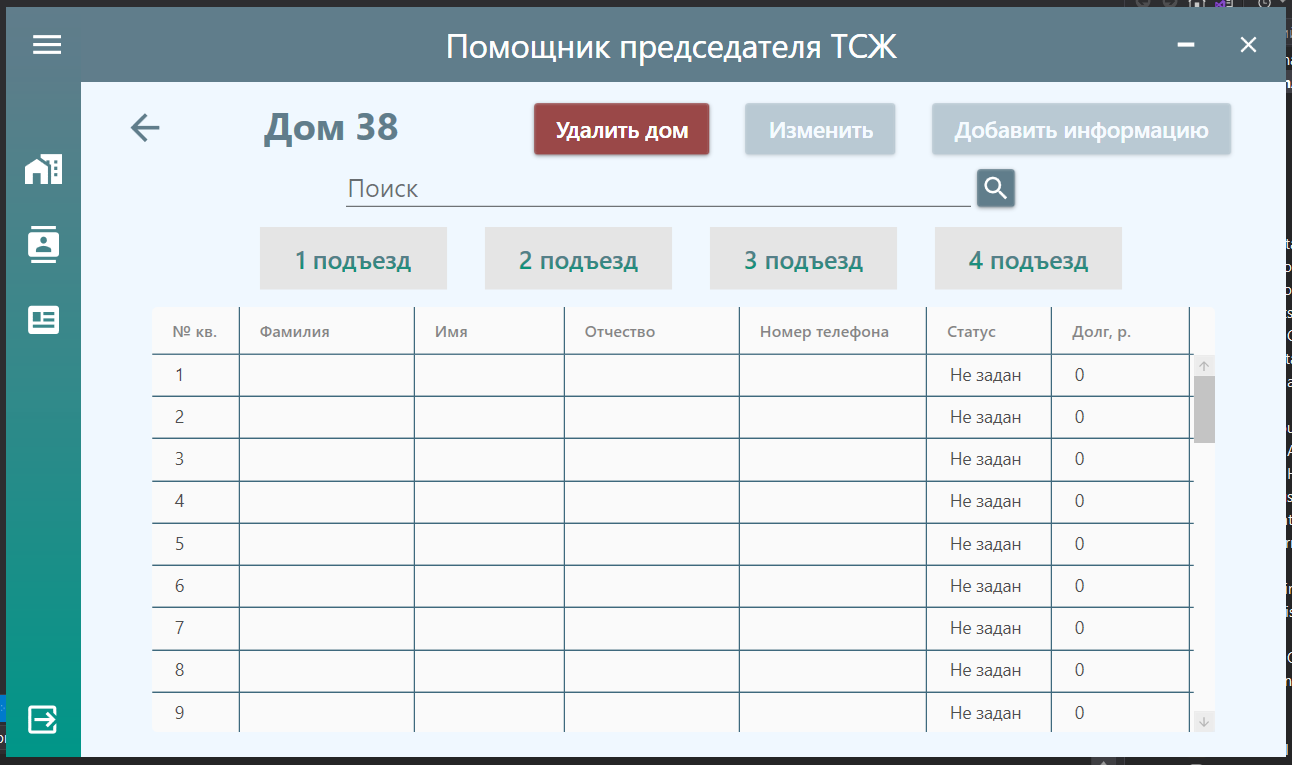


Рисунок 6.7 – Страница просмотра квартир после нажатия на кнопку «Подъезд»

При нажатии на строку с данными появляется возможность добавить жильца.

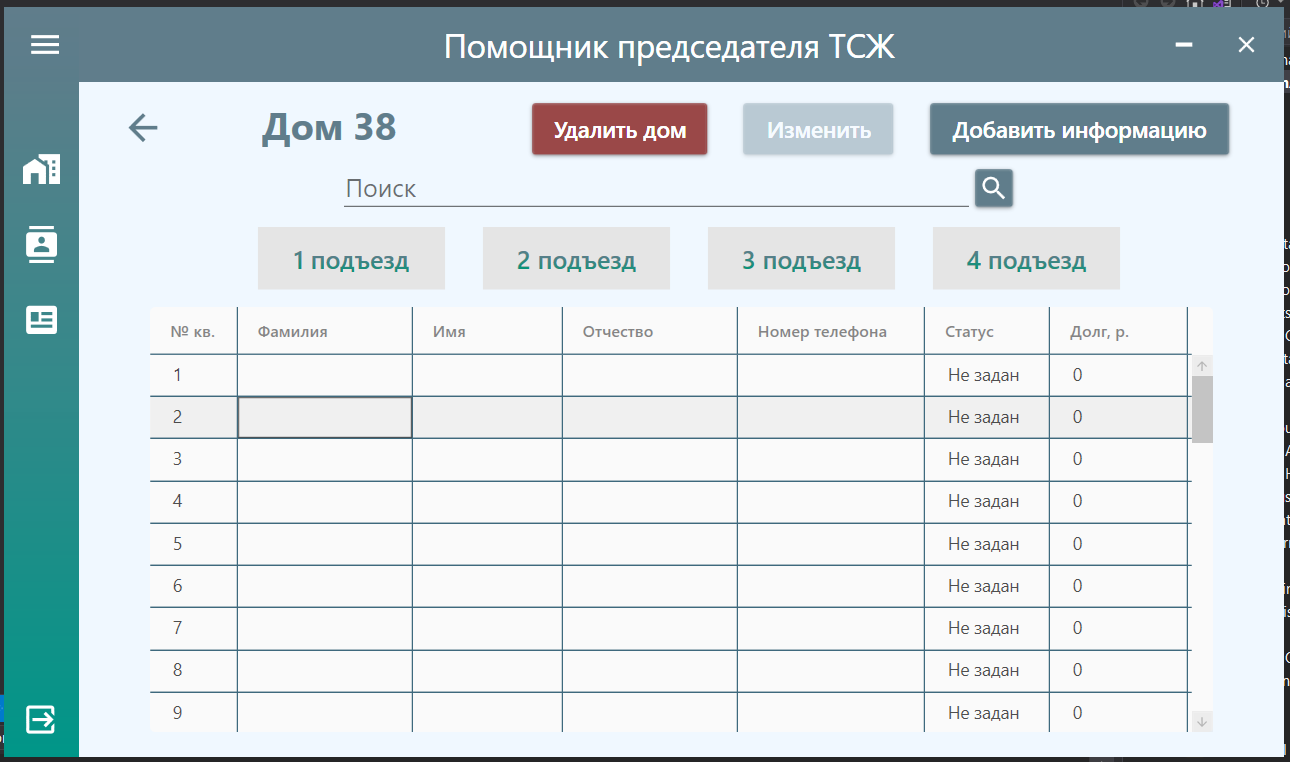


Рисунок 6.7 – Активация кнопки «Добавить информацию»

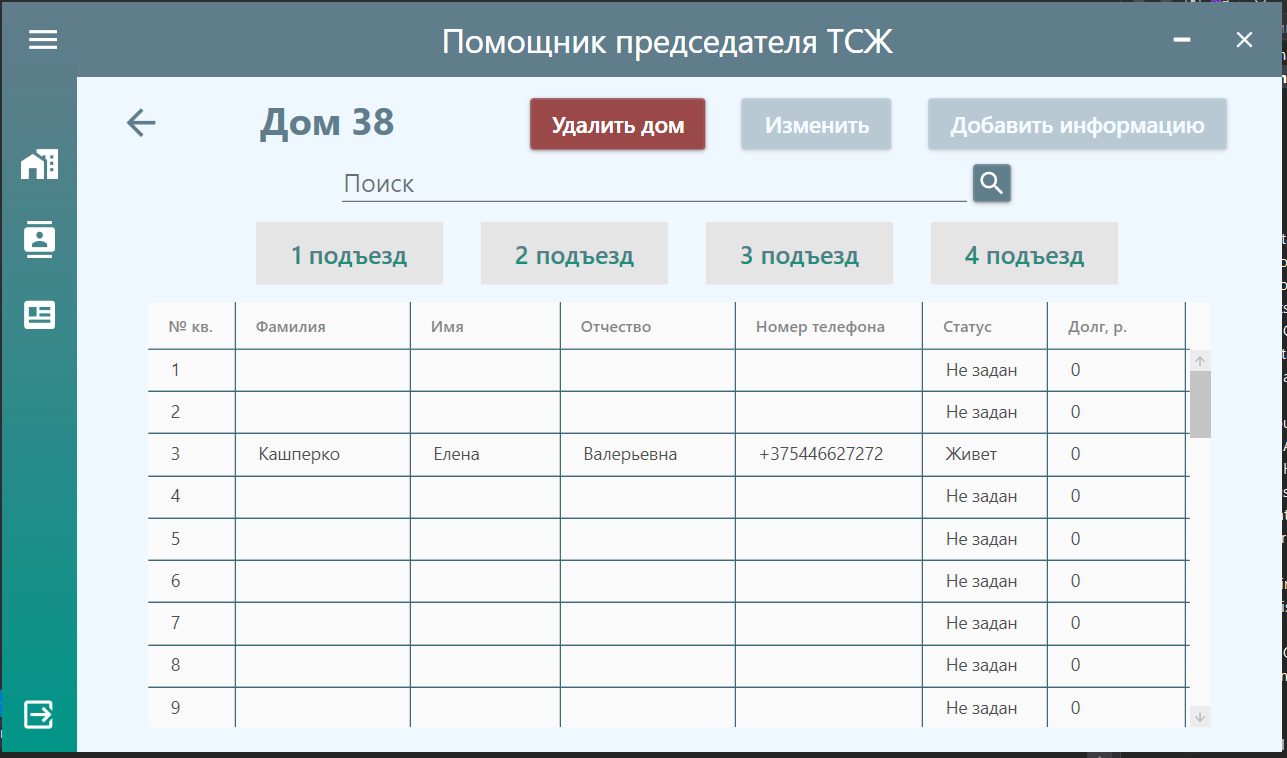


Рисунок 6.8 – Добавленный собственник в таблице

При нажатии на уже заполненную строку, подсвечивается лишь кнопка «Изменить». Нажав ее, можно изменить информацию о выбранном собственнике или удалить его.

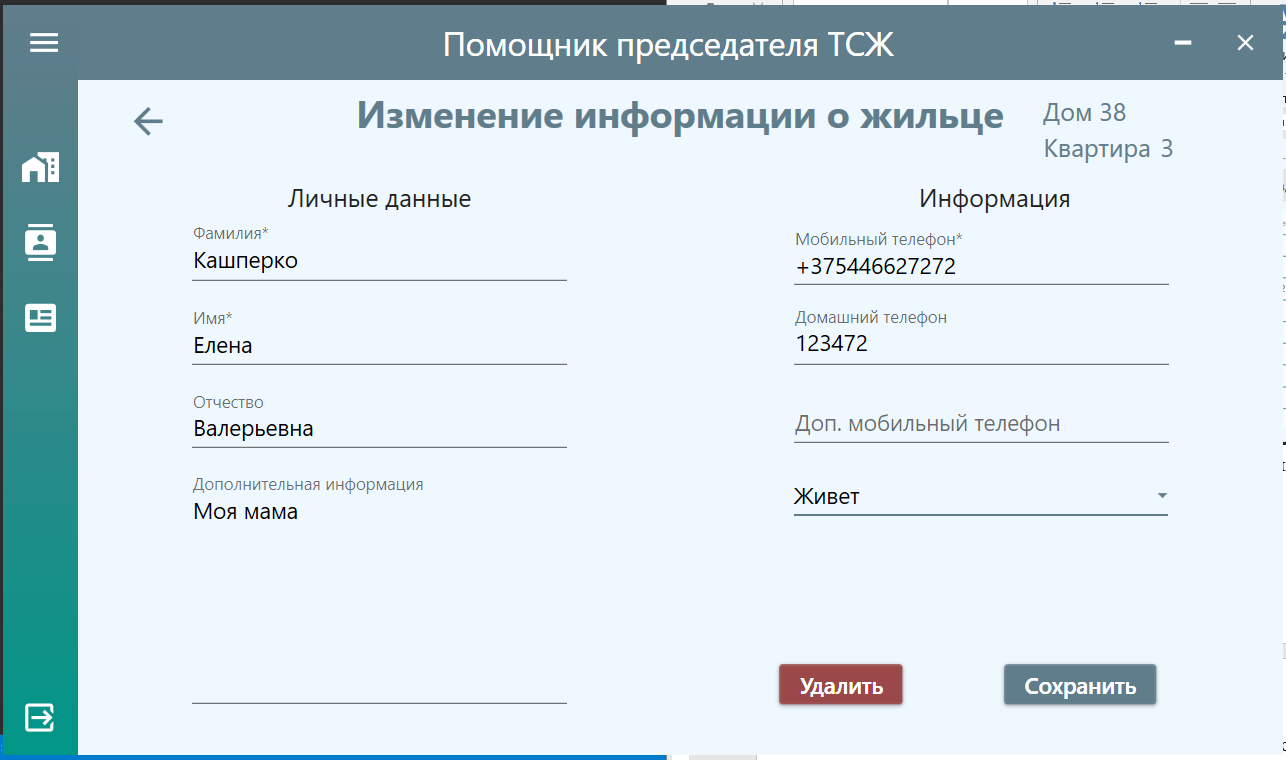


Рисунок 6.9 – Изменение информации о жильце

Также можно удалить дом, если два раза нажать по кнопке.

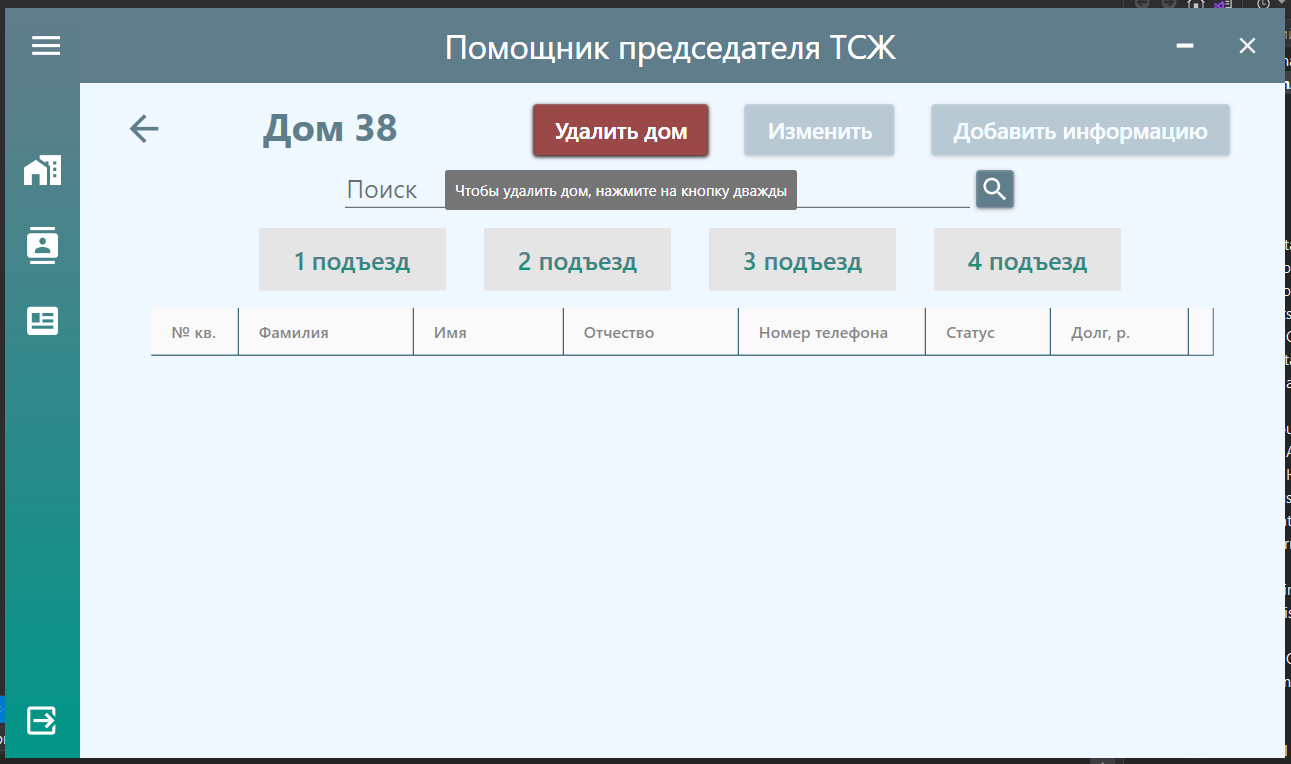


Рисунок 6.10 – Возможность удаления дома

Перейдя к разделу «Записная книжка», если она не заполнена, она будет выглядеть так:

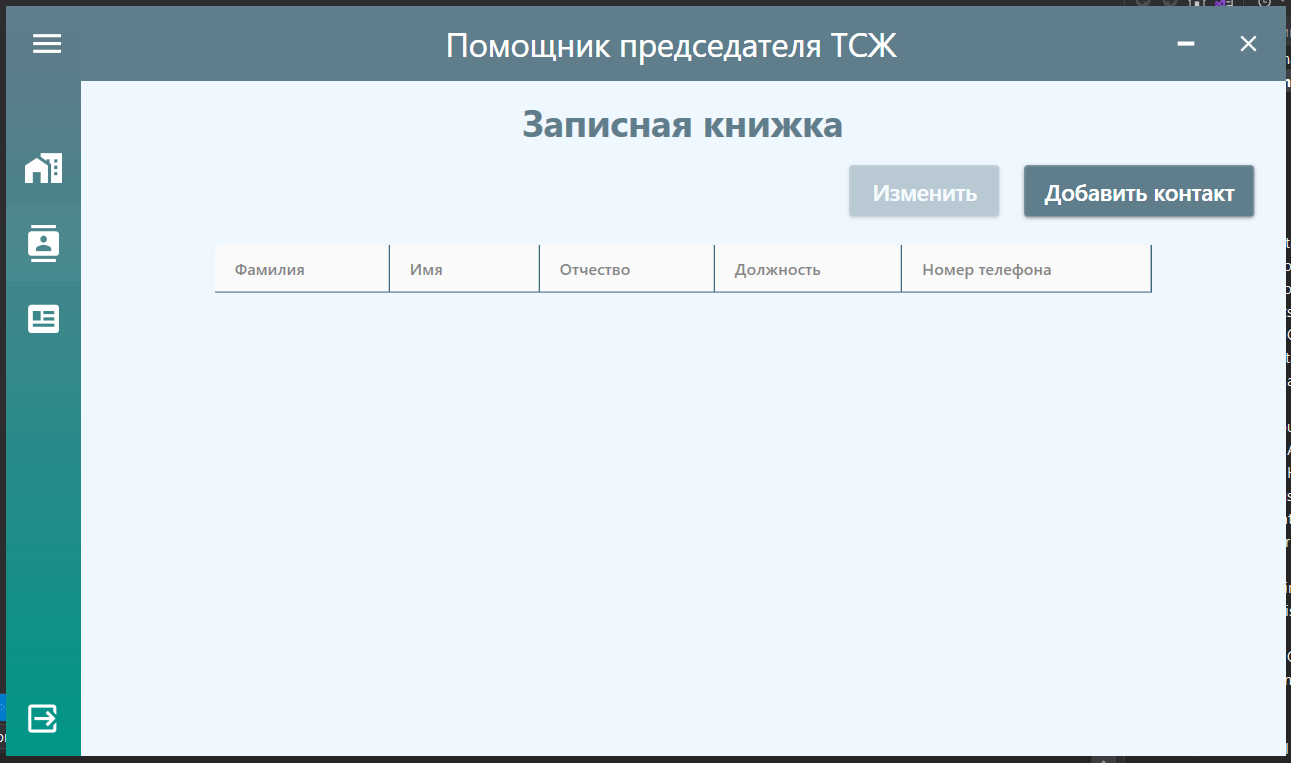


Рисунок 6.11 – Страница «Записная книжка», если в ней нет номеров

Аналогично добавлению дома или жильца, заполняется форма:

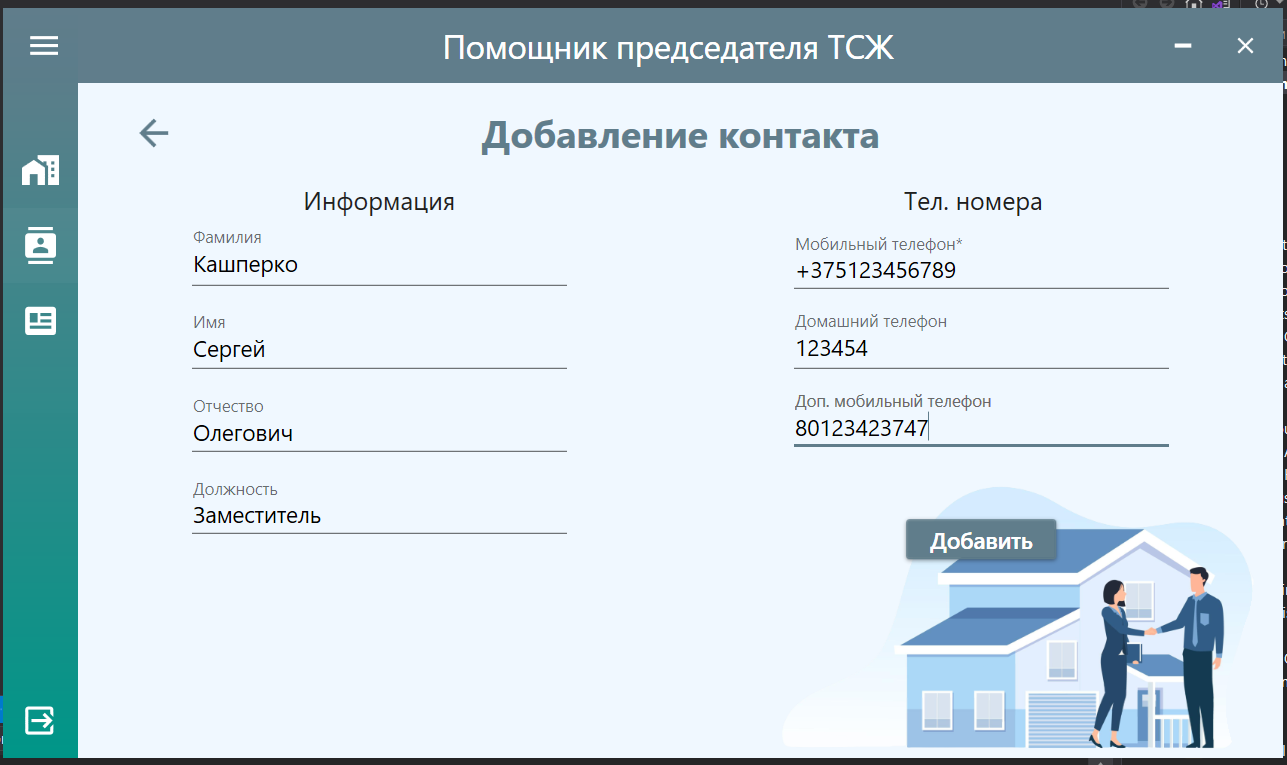


Рисунок 6.11 – Страница добавления контакта

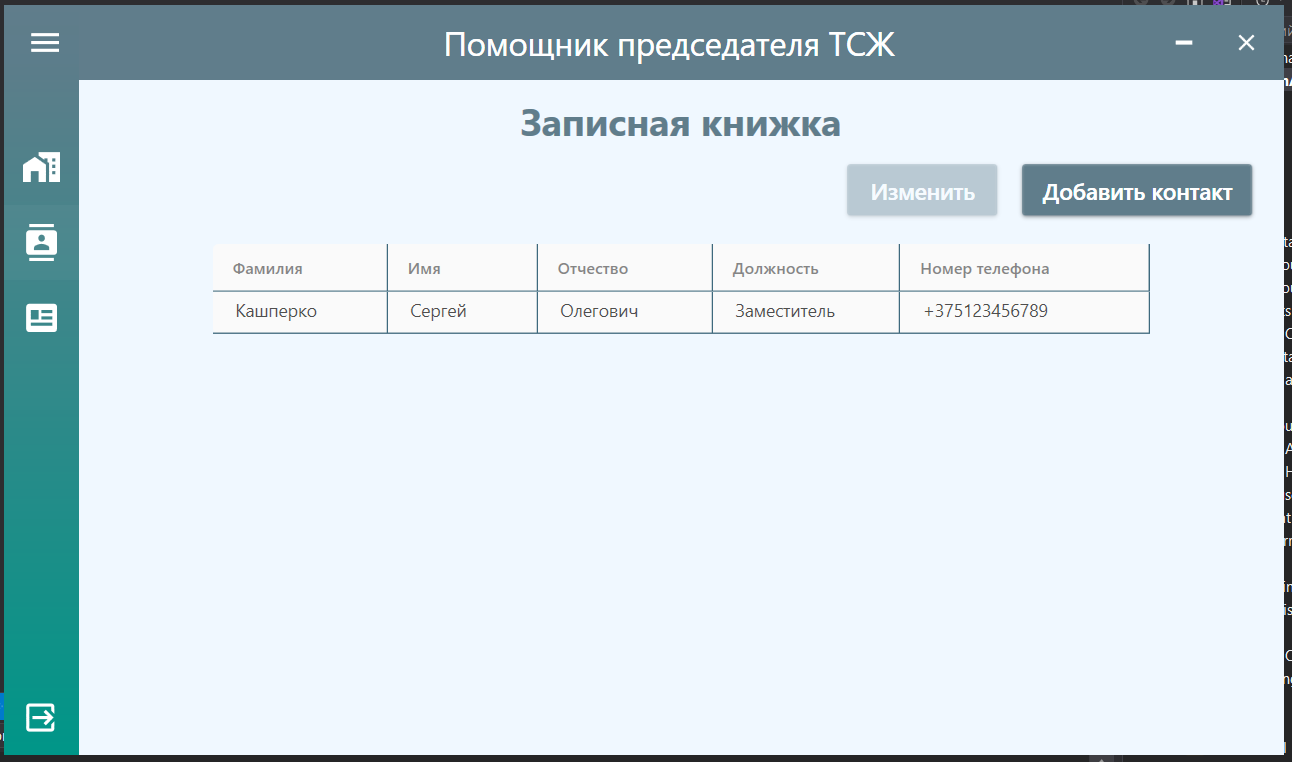


Рисунок 6.12 – Страница «Записная книжка»

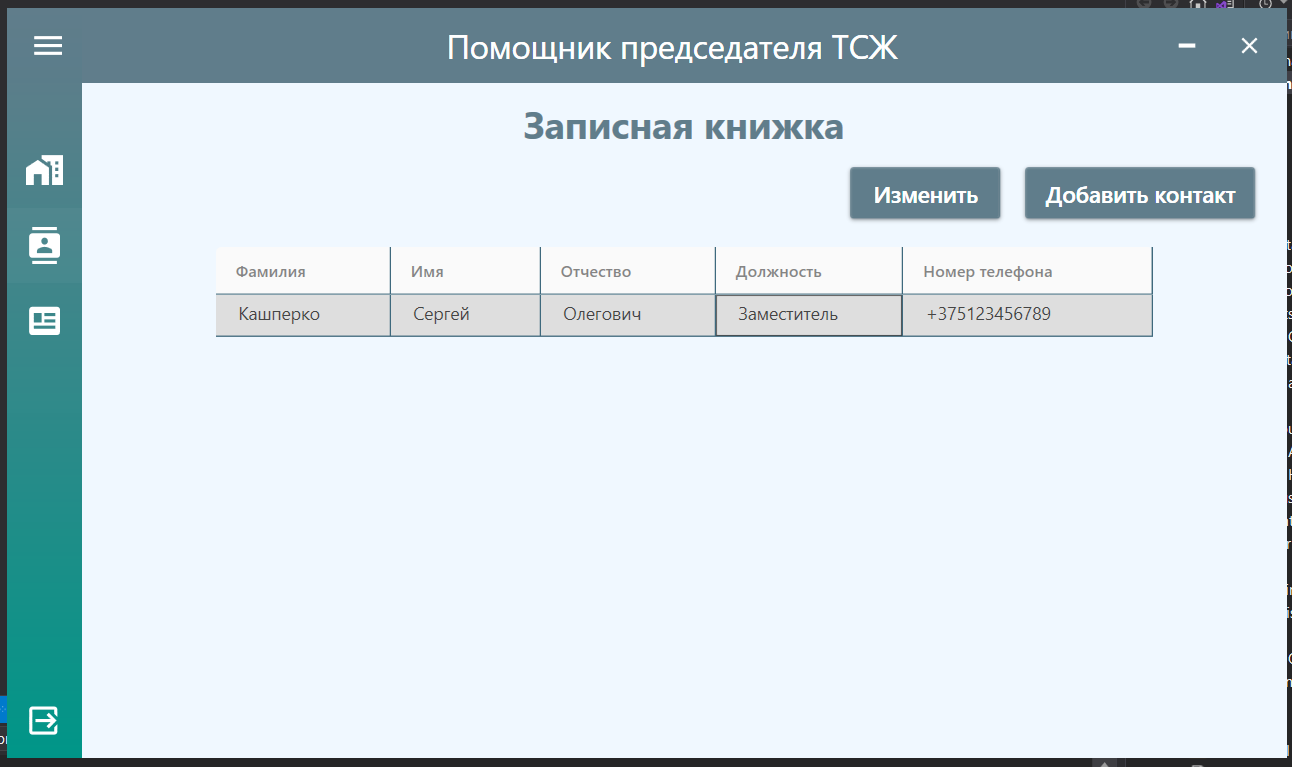


Рисунок 6.13 – Страница «Записная книжка» при выделении контакта

При нажатии на раздел «Справка» открывается данная страница:

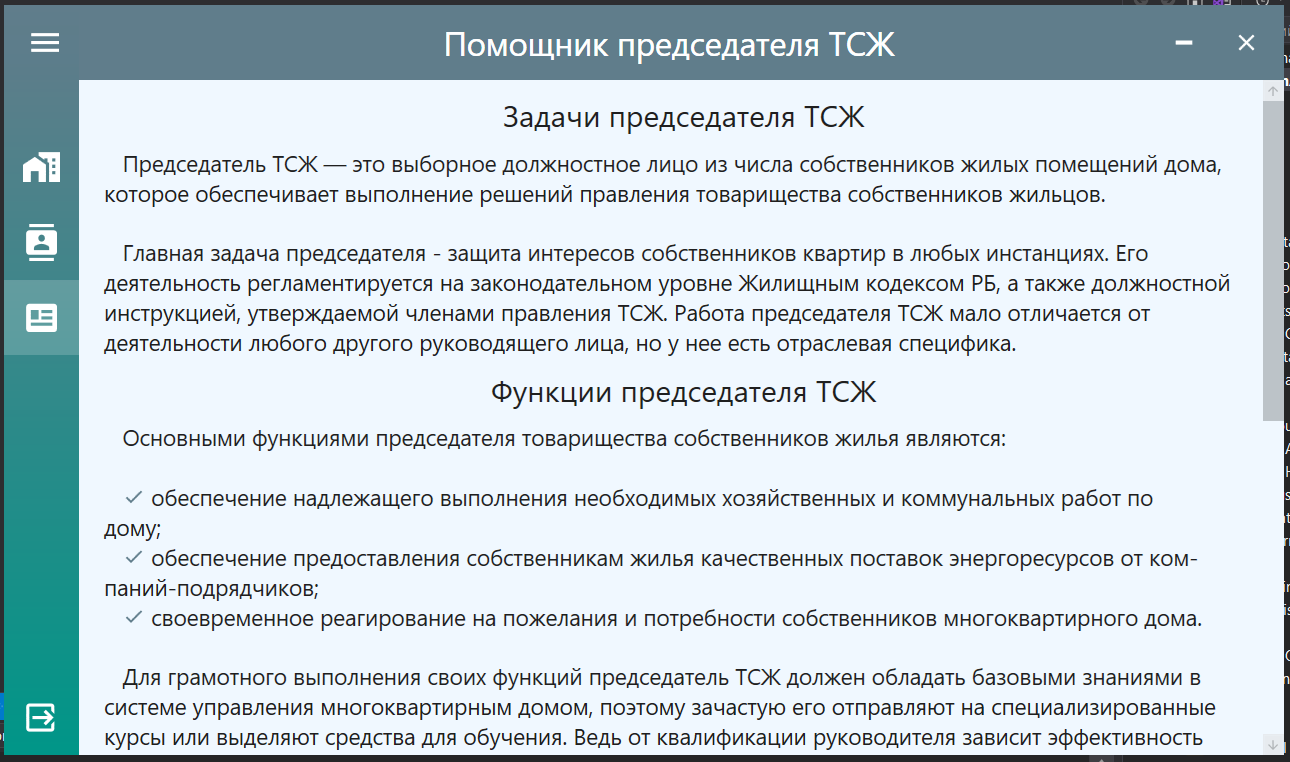


Рисунок 6.14 – Страница «Справка»

В том случае, когда заходит бухгалтер на свой аккаунт, он видит на главной странице дома, доступные председателю, за которым он закреплен.

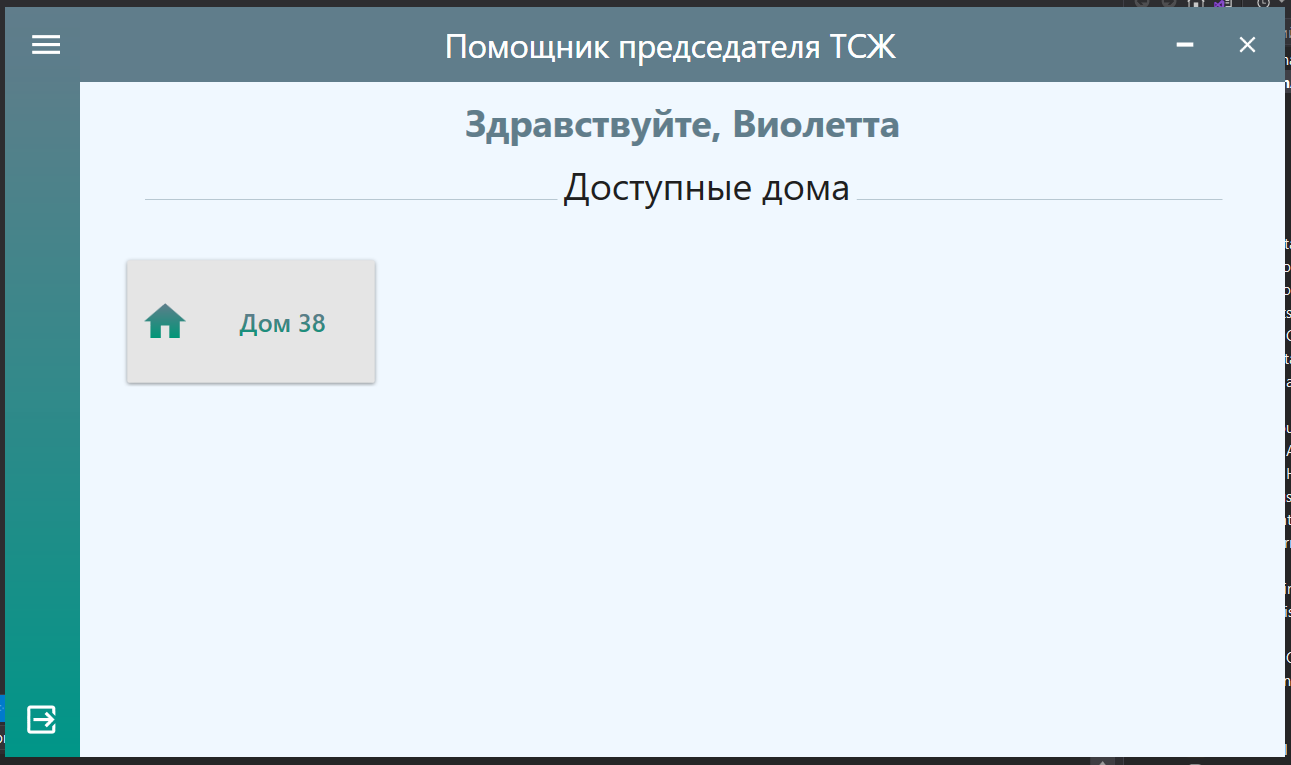


Рисунок 6.14 – Страница «Справка»

Нажав на кнопку, он перейдет на страницу, на которой сможет только менять данные о жильцах, в том числе и их текущую задолженность (в отличии от председателя).

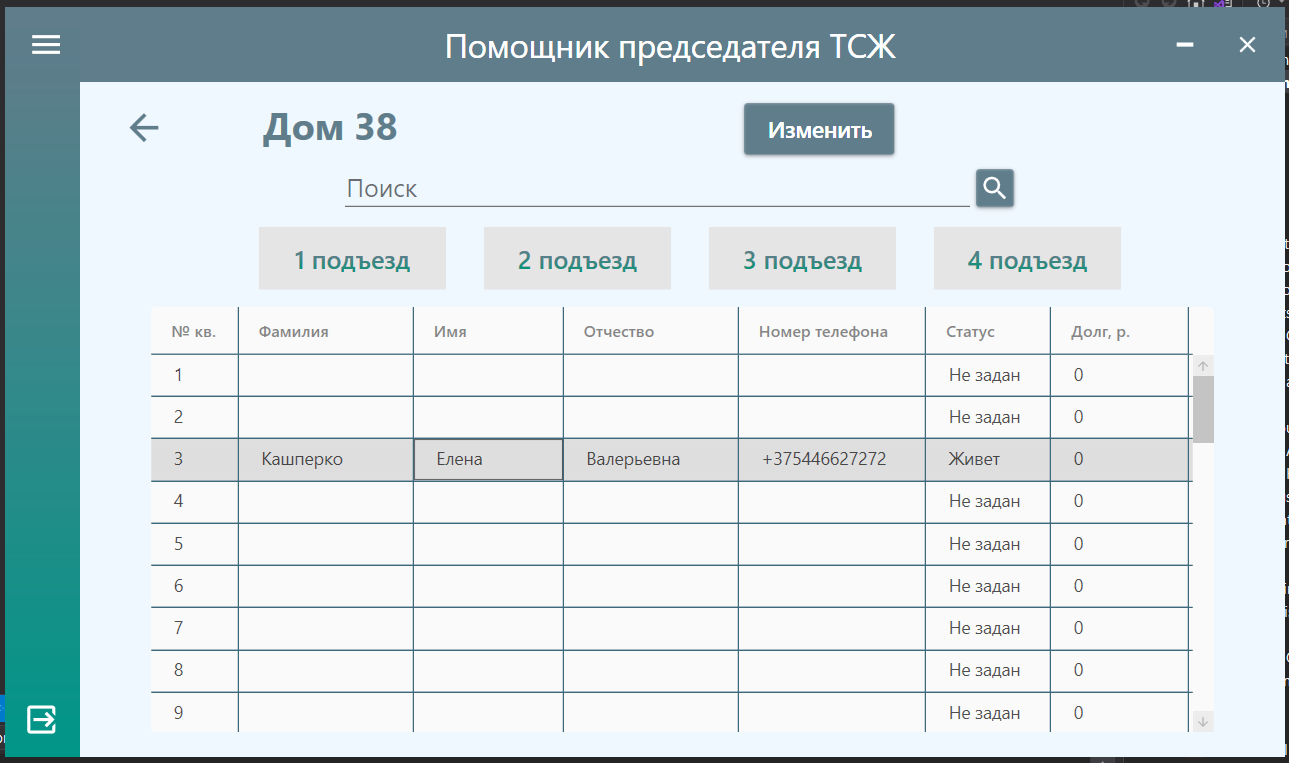


Рисунок 6.15 – Страница «Справка»

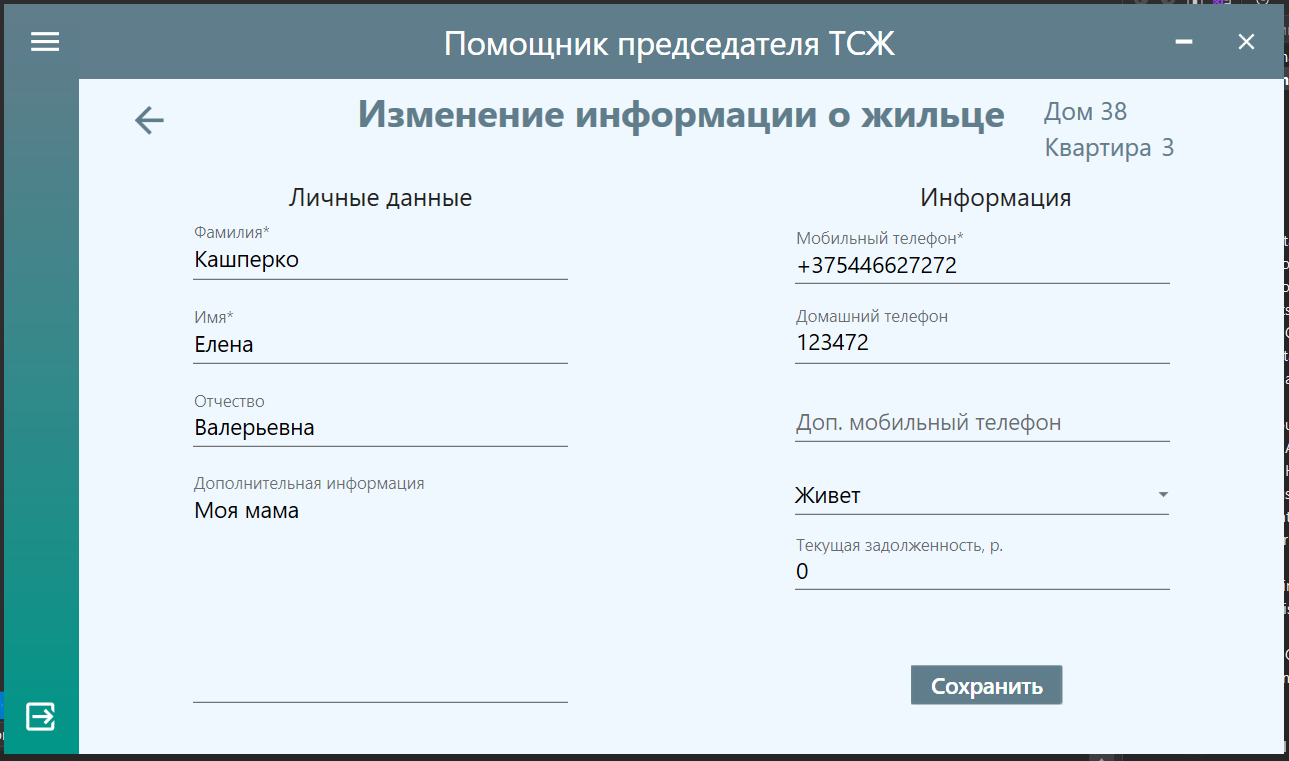


Рисунок 6.16 – Страница «Изменение информации о жильце» (для бухгалтера)

ЗАКЛЮЧЕНИЕ

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Помощник председателя товарищества собственников жилья». При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения.

В программном средстве были реализованы следующие функции председателя ТСЖ:

* + выполнять регистрацию и авторизацию;
  + поддерживать работу c базой данных;
  + добавлять и изменять информацию о жилых домах и собственниках квартир;
  + просматривать информацию о жилых домах и собственниках квартир;
  + выполнять поисковые запросы;
  + добавлять контакты работников дома и организаций в записную книжку;
  + выполнять регистрацию бухгалтера для определенного дома.

Функции бухгалтера ТСЖ:

* + выполнять авторизацию;
  + просматривать информацию о собственниках квартир;
  + добавлять и обновлять сведения о текущих задолженностях собственников.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

СПИСОК ЛИТЕРАТУРЫ

1. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.

2. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> . Дата доступа: 20.04.2020

3. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com> . Дата доступа: 20.04.2020

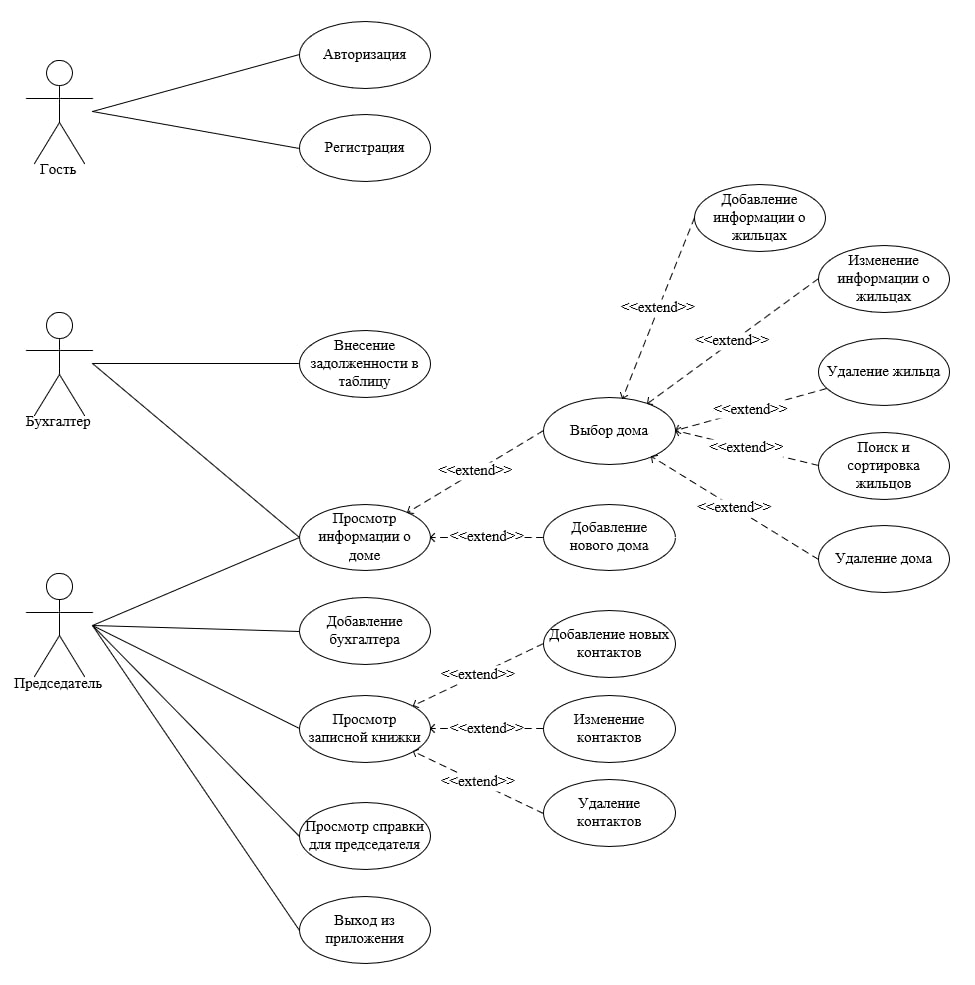
4. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru> Дата доступа: 13.04.2020

5 Microsoft Docs Archived Content [Электронный ресурс]/ Режим доступа: <https://docs.microsoft.com/en-us/archive/> Дата доступа: 26.04.2020

6 Форум для программистов или разработчиков [Электронный ресурс] – <https://stackoverflow.com/> – Дата доступа: 5.05.2020

**ПРИЛОЖЕНИЕ А**

Диаграмма использования



ПРИЛОЖЕНИЕ Б

Листинг 1. Код класса FrameNavigationService

using HOAChairmanAssistant.Model;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

namespace HOAChairmanAssistant.Helpers.Navigation

{

public class FrameNavigationService : IFrameNavigationService, INotifyPropertyChanged

{

#region Fields

private readonly Dictionary<string, Uri> \_pagesByKey;

private readonly List<string> \_historic;

private string \_currentPageKey;

#endregion

#region Properties

public string CurrentPageKey

{

get

{

return \_currentPageKey;

}

private set

{

if (\_currentPageKey == value)

{

return;

}

\_currentPageKey = value;

OnPropertyChanged("CurrentPageKey");

}

}

public object Parameter { get; private set; }

public object Parameter2 { get; private set; }

#endregion

#region Ctors and Methods

public FrameNavigationService()

{

\_pagesByKey = new Dictionary<string, Uri>();

\_historic = new List<string>();

}

public void GoBack()

{

if (\_historic.Count > 1)

{

\_historic.RemoveAt(\_historic.Count - 1);

NavigateTo(\_historic.Last(), null);

}/\*

else

{

MessageBox.Show("You are at Main Page");

}\*/

}

public void NavigateTo(string pageKey)

{

NavigateTo(pageKey, null);

}

public virtual void NavigateTo(string pageKey, object parameter)

{

lock (\_pagesByKey)

{

if (!\_pagesByKey.ContainsKey(pageKey))

{

throw new ArgumentException(string.Format("No such page: {0} ", pageKey), "pageKey");

}

Frame frame = GetDescendantFromName(Application.Current.Windows[0], "MainFrame") as Frame;

if (Application.Current.Windows.Count > 2)

{

if (Application.Current.Windows[2].Name == "MainWindow")

frame = GetDescendantFromName(Application.Current.Windows[2], "MainFrame") as Frame;

}

if (frame != null)

{

frame.Source = \_pagesByKey[pageKey];

}

Parameter = parameter;

\_historic.Add(pageKey);

CurrentPageKey = pageKey;

}

}

public virtual void NavigateTo(string pageKey, object parameter, object parameter2)

{

lock (\_pagesByKey)

{

if (!\_pagesByKey.ContainsKey(pageKey))

{

throw new ArgumentException(string.Format("No such page: {0} ", pageKey), "pageKey");

}

Frame frame = GetDescendantFromName(Application.Current.Windows[0], "MainFrame") as Frame;

if (Application.Current.Windows.Count > 2)

{

if (Application.Current.Windows[2].Name == "MainWindow")

frame = GetDescendantFromName(Application.Current.Windows[2], "MainFrame") as Frame;

}

if (frame != null)

{

frame.Source = \_pagesByKey[pageKey];

}

Parameter = parameter;

Parameter2 = parameter2;

\_historic.Add(pageKey);

CurrentPageKey = pageKey;

}

}

public void NavigateTo(string pageKey, House aboutHouse, Flat flat)

{

lock (\_pagesByKey)

{

if (!\_pagesByKey.ContainsKey(pageKey))

{

throw new ArgumentException(string.Format("No such page: {0} ", pageKey), "pageKey");

}

Frame frame = GetDescendantFromName(Application.Current.Windows[0], "MainFrame") as Frame;

if (Application.Current.Windows.Count > 2)

{

if (Application.Current.Windows[2].Name == "MainWindow")

frame = GetDescendantFromName(Application.Current.Windows[2], "MainFrame") as Frame;

}

if (frame != null)

{

frame.Source = \_pagesByKey[pageKey];

}

Parameter = aboutHouse;

Parameter2 = flat;

\_historic.Add(pageKey);

CurrentPageKey = pageKey;

}

}

public void Configure(string key, Uri pageType)

{

lock (\_pagesByKey)

{

if (\_pagesByKey.ContainsKey(key))

{

\_pagesByKey[key] = pageType;

}

else

{

\_pagesByKey.Add(key, pageType);

}

}

}

private static FrameworkElement GetDescendantFromName(DependencyObject parent, string name)

{

var count = VisualTreeHelper.GetChildrenCount(parent);

if (count < 1)

{

return null;

}

for (var i = 0; i < count; i++)

{

var frameworkElement = VisualTreeHelper.GetChild(parent, i) as FrameworkElement;

if (frameworkElement != null)

{

if (frameworkElement.Name == name)

{

return frameworkElement;

}

frameworkElement = GetDescendantFromName(frameworkElement, name);

if (frameworkElement != null)

{

return frameworkElement;

}

}

}

return null;

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));

}

#endregion

}

}

**ПРИЛОЖЕНИЕ В**

Листинг 2. Код класса RelayCommandParametr

public class RelayCommandParametr : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RelayCommandParametr(Action<object> execute, Func<object, bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return this.canExecute == null || this.canExecute(parameter);

}

public void Execute(object parameter)

{

this.execute(parameter);

}

}

**ПРИЛОЖЕНИЕ Г**

Листинг 3. Код класса ViewModelLocator

using CommonServiceLocator;

using GalaSoft.MvvmLight.Ioc;

using HOAChairmanAssistant.Helpers.Navigation;

using HOAChairmanAssistant.ViewModel;

using System;

namespace HOAChairmanAssistant.Helpers.Locator

{

public class ViewModelLocator

{

static ViewModelLocator()

{

ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

SimpleIoc.Default.Register<LoginWindowViewModel>();

SimpleIoc.Default.Register<MainWindowViewModel>();

SetupNavigation();

}

private static void SetupNavigation()

{

var navigationService = new FrameNavigationService();

navigationService.Configure("Login", new Uri("../Pages/Login/LoginPage.xaml", UriKind.Relative));

navigationService.Configure("Registration", new Uri("../Pages/Login/RegistrationPage.xaml", UriKind.Relative));

navigationService.Configure("Houses", new Uri("../Pages/Houses/HousesPage.xaml", UriKind.Relative));

navigationService.Configure("AddHousePage", new Uri("../Pages/Houses/AddHousePage.xaml", UriKind.Relative));

navigationService.Configure("Information", new Uri("../Pages/Information/InformationPage.xaml", UriKind.Relative));

navigationService.Configure("Contacts", new Uri("../Pages/Contacts/ContactsPage.xaml", UriKind.Relative));

navigationService.Configure("AddContact", new Uri("../Pages/Contacts/AddContactPage.xaml", UriKind.Relative));

navigationService.Configure("UpdateContact", new Uri("../Pages/Contacts/UpdateContactPage.xaml", UriKind.Relative));

navigationService.Configure("AboutHouse", new Uri("../Pages/Houses/AboutHousePage.xaml", UriKind.Relative));

navigationService.Configure("AddOwnerPage", new Uri("../Pages/Owners/AddOwnerPage.xaml", UriKind.Relative));

navigationService.Configure("ChangeOwnerPage", new Uri("../Pages/Owners/ChangeOwnerPage.xaml", UriKind.Relative));

navigationService.Configure("AddAccountantPage", new Uri("../Pages/Houses/AddAccountantPage.xaml", UriKind.Relative));

navigationService.Configure("AccountantHomePage", new Uri("../Pages/Accountant/AccountantHomePage.xaml", UriKind.Relative));

navigationService.Configure("AccountantChangeDebt", new Uri("../Pages/Accountant/AccountantChangeDebt.xaml", UriKind.Relative));

SimpleIoc.Default.Register<IFrameNavigationService>(() => navigationService);

}

public MainWindowViewModel Main

{

get

{

return ServiceLocator.Current.GetInstance<MainWindowViewModel>();

}

}

public LoginWindowViewModel LoginWindowViewModel

{

get

{

return ServiceLocator.Current.GetInstance<LoginWindowViewModel>();

}

}

public static void Cleanup()

{

}

}

}

**ПРИЛОЖЕНИЕ Д**

Листинг 4. Код класса MainViewModel

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using GalaSoft.MvvmLight.Threading;

using HOAChairmanAssistant.DataLayer.EF;

using HOAChairmanAssistant.Helpers.GlobalData;

using HOAChairmanAssistant.Helpers.MessageWindow;

using HOAChairmanAssistant.Helpers.Navigation;

using HOAChairmanAssistant.Model;

namespace HOAChairmanAssistant.ViewModel

{

public class MainWindowViewModel : ViewModelBase

{

#region Private Fields

private User user;

private HOAChairmanAssistantContext context = new HOAChairmanAssistantContext();

private bool isAccountant = false;

private bool isChairman = false;

private bool isOpenDialog;

private IFrameNavigationService \_navigationService;

#endregion

#region Public Fields

public User User

{

get

{

return user;

}

set

{

if (user == value)

{

return;

}

user = value;

RaisePropertyChanged();

}

}

public bool IsOpenDialog

{

get

{

return isOpenDialog;

}

set

{

if (isOpenDialog == value)

{

return;

}

isOpenDialog = value;

RaisePropertyChanged();

}

}

public bool IsAccountant

{

get

{

return isAccountant;

}

set

{

if (isAccountant == value)

{

return;

}

isAccountant = value;

RaisePropertyChanged();

}

}

public bool IsChairman

{

get

{

return isChairman;

}

set

{

if (isChairman == value)

{

return;

}

isChairman = value;

RaisePropertyChanged();

}

}

public string Message { get; internal set; }

#endregion

#region Commands

private RelayCommand \_loginpageCommand;

public RelayCommand LoginPageCommand

{

get

{

return \_loginpageCommand

?? (\_loginpageCommand = new RelayCommand(

() =>

{

DispatcherHelper.CheckBeginInvokeOnUI(

() =>

{

\_navigationService.NavigateTo("Login");

});

}));

}

}

private RelayCommand \_housesCommand;

public RelayCommand HousesCommand

{

get

{

return \_housesCommand

?? (\_housesCommand = new RelayCommand(

() =>

{

if (GlobalData.UserId != 0)

{

\_navigationService.NavigateTo("Houses");

var user = context.Users.Where(x => x.UserId == GlobalData.UserId).FirstOrDefault();

var userAcc = context.Users.Where(y => y.UserId == user.AccountantId).FirstOrDefault();

if (userAcc != null)

{

isAccountant = true;

}

else

{

isAccountant = false;

}

var userChairman = context.Users.Where(y => y.UserId == user.AccountantId).FirstOrDefault();

if (userChairman == null)

{

isChairman = true;

}

else

{

isChairman = false;

}

}

}));

}

}

private RelayCommand \_informationCommand;

public RelayCommand InformationCommand

{

get

{

return \_informationCommand

?? (\_informationCommand = new RelayCommand(

() =>

{

\_navigationService.NavigateTo("Information");

}));

}

}

private RelayCommand \_contactsCommand;

public RelayCommand ContactsCommand

{

get

{

return \_contactsCommand

?? (\_contactsCommand = new RelayCommand(

() =>

{

\_navigationService.NavigateTo("Contacts");

}));

}

}

private RelayCommand \_settingsCommand;

public RelayCommand SettingsCommand

{

get

{

return \_settingsCommand

?? (\_settingsCommand = new RelayCommand(

() =>

{

\_navigationService.NavigateTo("Settings");

}));

}

}

private RelayCommand closeDialodCommand;

public RelayCommand CloseDialodCommand

{

get

{

return closeDialodCommand

?? (closeDialodCommand = new RelayCommand(

() =>

{

IsOpenDialog = false;

}));

}

}

#endregion

#region ctor

public MainWindowViewModel(IFrameNavigationService navigationService)

{

\_navigationService = navigationService;

}

#endregion

}

}

**ПРИЛОЖЕНИЕ Е**

Листинг 5. Код класса HOAChairmanAssistantContext

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.Entity;

using HOAChairmanAssistant.Model;

namespace HOAChairmanAssistant.DataLayer.EF

{

public class HOAChairmanAssistantContext : DbContext

{

public HOAChairmanAssistantContext()

: base("name=HOAChairmanAssistantContext")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

// Отражение таблиц базы данных на свойства с типом DbSet

public DbSet<User> Users { get; set; }

public DbSet<Address> Addresses { get; set; }

public DbSet<House> Houses { get; set; }

public DbSet<Porch> Porches { get; set; }

public DbSet<Flat> Flats { get; set; }

public DbSet<PhoneNumber> PhoneNumbers { get; set; }

public DbSet<Owner> Owners { get; set; }

public DbSet<Contact> Contacts { get; set; }

}

}