МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

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

Тема Программное средство «Управление интернами»

Исполнитель

студент 2 курса группы 4 Городилов Михаил Петрович

(Ф.И.О.)

Руководитель работы ассистент Рауба А.А.

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

Курсовой проект защищен с оценкой

Председатель Рауба А.А.

(подпись)

Минск 2020

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc41244345)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc41244346)

[2 Анализ требований к программному средству и разработка функциональных требований 6](#_Toc41244347)

[3 Проектирование программного средства 8](#_Toc41244348)

[3.1 Структуры и взаимоотношения между классами 8](#_Toc41244349)

[3.2 Описание данных 12](#_Toc41244350)

[4 Реализация программного средства 15](#_Toc41244351)

[4.1 Реализация классов для работы с данными 15](#_Toc41244352)

[4.2 Реализация классов шифрования 18](#_Toc41244353)

[4.3 Реализация вспомогательных классов 19](#_Toc41244354)

[4.4 Реализация классов интерфейса 22](#_Toc41244355)

[4.5 Сборка проекта 30](#_Toc41244356)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 31](#_Toc41244357)

[5.1 Тестирование страницы авторизации 31](#_Toc41244358)

[5.2 Тестирование страницы регистрации 32](#_Toc41244359)

[5.3 Тестирование страницы не проверенных пользователей 33](#_Toc41244360)

[5.4 Тестирование страницы менеджера 34](#_Toc41244361)

[5.5 Тестирование страницы интерна 35](#_Toc41244362)

[6 Руководство по установке и использованию 39](#_Toc41244363)

[6.1 Системные требования 39](#_Toc41244364)

[6.2 Установка 39](#_Toc41244365)

[ЗАКЛЮЧЕНИЕ 41](#_Toc41244366)

[Список использованных источников 42](#_Toc41244367)

[ПРИЛОЖЕНИЕ А 43](#_Toc41244368)

[ПРИЛОЖЕНИЕ Б 45](#_Toc41244369)

[ПРИЛОЖЕНИЕ В 51](#_Toc41244370)

[ПРИЛОЖЕНИЕ Г 53](#_Toc41244371)

ВВЕДЕНИЕ

Приложение «Управление интернами» – это рабочий инструмент для организации стажировок в компании. В частности, крупные IT компании имеют потребность в подобном ПО, которое позволило бы автоматизировать процессы отслеживания стажировок.

Цель данного курсового проекта – проектирование и разработка данного приложения.

При выполнении курсового проекта будут использованы принципы и приемы ООП. Для разработки графической составляющей данного продукта используется технология Windows Presentation Foundation (WPF).

В первом разделе представлен анализ прототипов и литературных источников.

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

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

На сегодняшний день можно встретить очень малое количество доступных программных решений, разработанных для управления стажировками, т.к. в основном компании имеют отдельное ПО для учёта сотрудников, без функционала для интернов, т.к. стажировки в IT компаниях раньше не были востребованы. За неимением аналогов специализированного ПО, рассмотрим программу «Connecteam», рассчитанную на сотрудников [1]:

«Connecteam» ⎯ это ПО для администрирования персонала. Программа может быть использована администраторами и самими сотрудниками.



Рисунок 1.1 – Главное окно ПС Connecteam

Проанализировав часть программы, рассчитанную на повышение квалификации сотрудников, можно выделить её основные плюсы и минусы.

Основные плюсы:

* Возможность создавать базу знаний компании;
* Наличие интерактивных опросов для проверки знаний;
* Возможность получить результаты развития сотрудников в графиках.

Основные минусы:

* Обобщённость продукта по сферам деятельности;
* Нет возможности ограничить доступность для интернов;
* Отсутствие реализации стажировок как таковых;

На основе проведенного анализа и с учетом требований, указанных в задании

на курсовое проектирование, сформированы следующие требования:

* Разработать бизнес логику приложения;
* Реализовать программный интерфейс;
* Реализовать нотификацию пользователей.

Для разработки данного ПС были выбраны следующие средства разработки:

* Интегрированная среда разработки Microsoft Visual Studio 2017;
* Программная платформа .NET Framework 4.6;
* Язык программирования C#;
* Технология Windows Presentation Foundation (WPF);
* Расширяемый язык разметки XAML;
* Технология ADO.NET Entity Framework.

2 Анализ требований к программному средству и разработка функциональных требований

Описание функциональности ПС:

* Возможность авторизации и регистрации;
* Наличие интерфейса для управления заявками на создание аккаунта;
* Возможность просматривать список стажировок и поиск по нему;
* Возможность менять картинку профиля;
* Возможность выставления временных рамок стажировки;
* Наличие интерфейса для частичного редактирования стажировки;
* Возможность назначать встречу со студентом;
* Возможность завершения стажировки;
* Наличие оповещения в программе;
* Наличие оповещения через email;

Вариант использования: в компании появляется нужда в интернах, данное ПС могло бы предоставить необходимые для менеджера и интерна данные о стажировке, упростило работу менеджеров с интернами, если основное ПО компании сложно масштабировать под интернатуры.

Спецификация функциональных требований:

1. Авторизация должна быть безопасна для пользователей, т.е. пароли будут храниться в БД в зашифрованном виде;
2. Основная часть функций будет реализовывать работу с БД, чтобы данные функции не блокировали пользовательский интерфейс при отработке, требуется сделать их асинхронными;
3. Функциональность оповещений по Email будет нуждаться в секретной информации (логин и пароль для smtp сервера), поэтому данная информация будет храниться в конфигурационных файлах;

Ниже, а также в графических материалах приведена диаграмма использования ПС (Use Case).

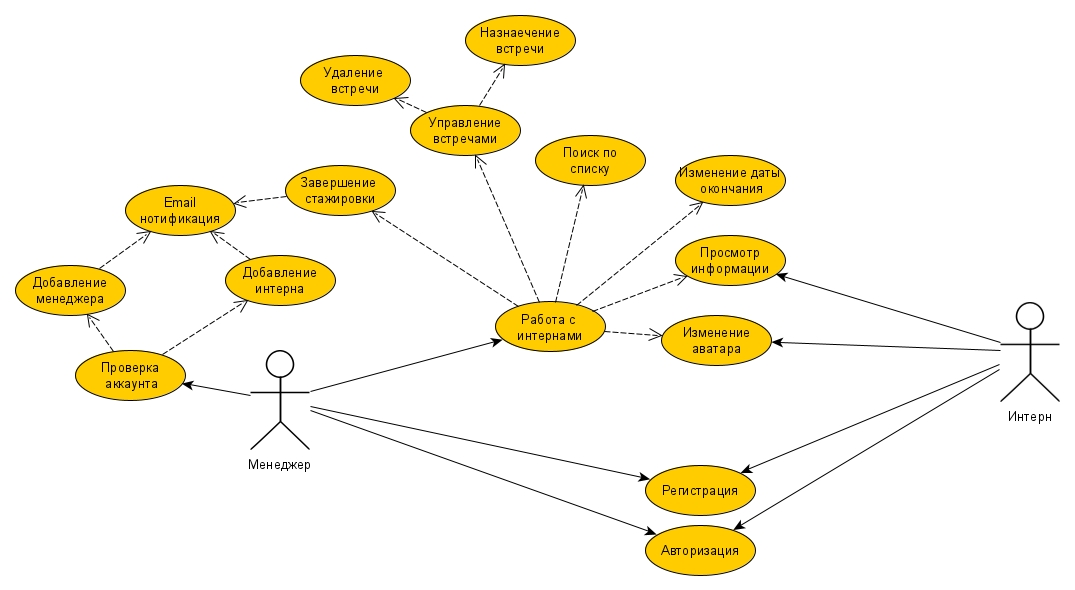


Рисунок 2.1 – Диаграмма использования

На данной диаграмме приведены возможные применения программы для таких ролей как «менеджер» и «интерн», далее будет описана реализация этих use case’ов.

3 Проектирование программного средства

3.1 Структуры и взаимоотношения между классами

Приложение построено на основе паттерна MVVM. Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения [2].



Рисунок 3.1 – Визуальное представление MVVM

Ниже приведены диаграммы классов ПС. Более подробные диаграммы приведены в графических материалах.

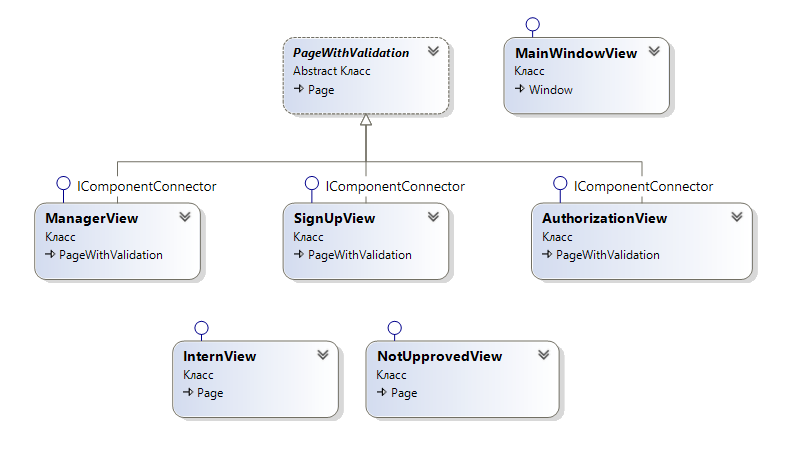


Рисунок 3.2 – Диаграмма классов представлений

Класс MainView наследуется от встроенного класса Window и представляет основное окно приложения.

Остальные классы наследуются от встроенного класса Page, который представляет страницы приложения, либо от реализованного класса PageWithValidation, который служит для валидации полей страницы.

Класс AuthorizationView реализует страницу авторицации.

Класс SignUpView реализует страницу регистрации.

Класс NotUpprovedView реализует страницу для неподтвержённых пользователей.

Класс ManagerView реализует страницу администрирования.

Класс InternView реализует страницу конкретного интерна.

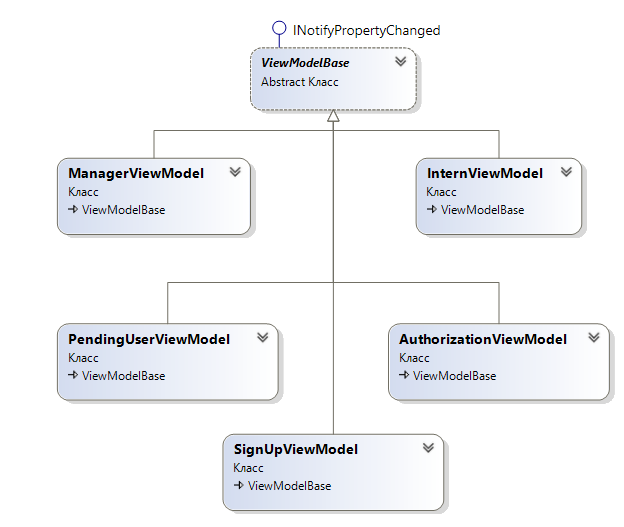


Рисунок 3.3 – Диаграмма классов моделей представлений

Класс ViewModelBase реализует интерфейс INotifyPropertyChanged, и является родительским для каждого класса моделей представлений.

Класс AuthorizationViewModel реализует логику авторицазии пользователя приложения.

Класс SignUpViewModel реализует логику регистрации пользователя.

Класс ManagerViewModel реализует логику управления интернами: добавление пользователей, поиск по списку интернов, переход на страницу интерна.

Класс PendingUserViewModel реализует модель представления пользователя, ожидающего подтверждение аккаунта.

Класс InternViewModel реализует логику для страницы интерна: изменение даты окончания, окончание, назначение встреч.

Класс NotUpprovedViewModel реализует единственную функцию – возврат на страницу авторизации.

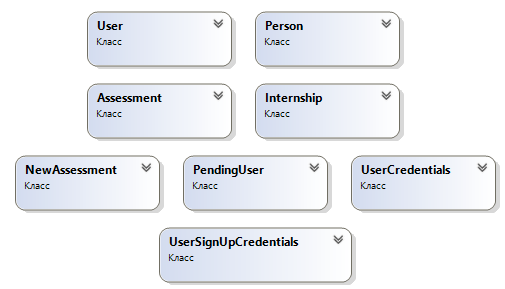


Рисунок 3.4 – Диаграмма классов моделей

Классы User, Person, Assessment, Internship являются представлениями Entity Framework, но их не достаточно, поэтому введены дополнительные классы для некоторых моделей представления. Подробное содержание классов так же находится в графических материалах.

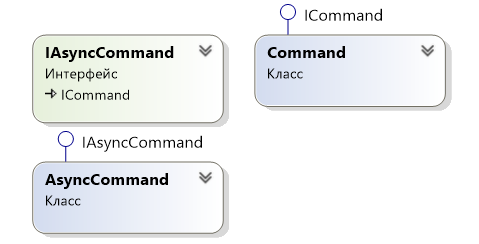


Рисунок 3.5 – Диаграмма классов моделей

Интерфейс ICommand является встроенным, требует реализации ивента CanExecuteChanged, и функций CanExecute, Execute. Т.е. паттерна комманда [3].

Интерфейс IAsyncCommand наследуется от ICommand и добавляет метод ExecuteAsync.

Класс Command реализует интерфейс ICommand.

Класс AsyncCommand реализует интерфейс IAsyncCommand и содержит обработку ошибок.

Ниже представленны вспомогательные классы приложения.

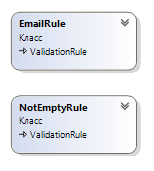


Рисунок 3.6 – Диаграмма классов валидации

Данные классы служат для валидации полей на таких страницах как SignUpView, AuthorizationView и ManagerView.



Рисунок 3.7 – Диаграмма классов валидации

Класс InternshipExtension содержит метод расширения GetSearchData для класса Internship.

Класс PasswordEncoder содержит логику хеширования и проверки пароля.

Класс AppNavHelper содержит методы, для отображения загрузочной панели, поля для хранения текущего пользователя и агрегирует NavigationService класса MainWindowView, для поддержки переходов на других страницах.

Класс ImageEncodingHelper реализует конвертацию png картинки в массив байт, для сохранения ранее упомянутой картинки в базе данных.

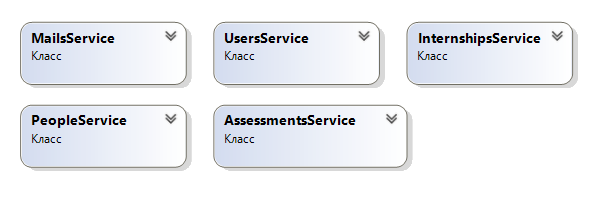


Рисунок 3.8 – Диаграмма классов сервисов

Класс MailsService содержит метод для отправки email сообщений.

Остальные классы представляют собой репозитории по работе с классом DataBaseContext и реализую методы добавления, редактирования, удаления и чтения.

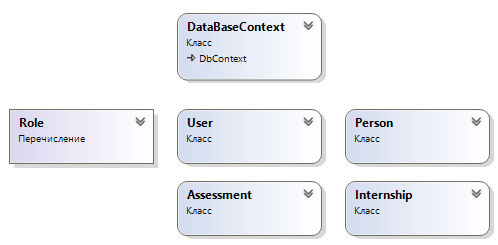


Рисунок 3.9 – Диаграмма классов Entity Framework

Класс DataBaseContext реализует паттерн UnitOfWork (содержит репозитории сущностей User, Person, Assessment, Internship), создаёт подключение к БД.

Перечисление Role композируется классом User.

3.2 Описание данных

Диаграмма базы данных сформированна в программе Microsoft SQL Server Management Studio 18 и представлена на рисунке 3.10. Для реализации необходимого функционала ПС и удобной разработки, в базе данных создано 5 таблиц: \_MigrationHistory – таблица созданная при включении миграций в Entity Framework, Users – таблица для хранения данных авторизации, People – хранение подробных данных о пользователях, Internships – таблица для хранения стажировок, Assessments – таблица для хранения встреч. Диаграмма БД приведена ниже, а также в графических материалах.

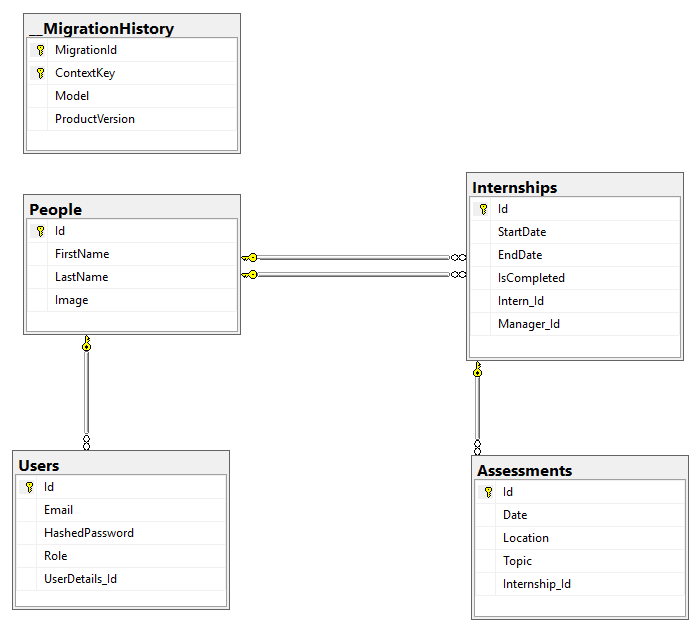


Рисунок 3.10 – Диаграмма базы данных

Таблица Users состоит из 5 столбцов:

* Идентификатора Id (первичный ключ);
* Электронная почта Email;
* Захешированный пароль HashedPassword;
* Роль пользователя Role;
* Внешнего ключа UserDetails\_Id.

Таблица People состоит из 4 столбцов:

* Идентификатора Id (первичный ключ);
* Имя FirstName;
* Фамилия LastName;
* Аватар Image.

Таблица Internships состоит из 6 столбцов:

* Идентификатора Id (первичный ключ);
* Даты начала StartDate;
* Даты окончания EndDate;
* Флага об окончании IsCompleted;
* Внешнего ключа Intern\_Id;
* Внешнего ключа Manager\_Id.

Таблица Assessments состоит из 5 столбцов:

* Идентификатора Id (первичный ключ);
* Даты и времени Date;
* Места Location;
* Темы Topic;
* Внешнего ключа Internship\_Id.

Идентификаторы генерируются автоматически (identity) [4].

Ниже приведена таблица преобразования типов данных.

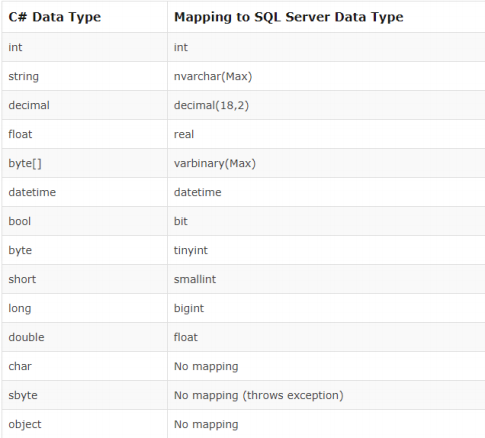


Рисунок 3.11 – Сопоставление типов

Все таблицы хранятся в базе данных на Microsoft SQL Server 2017, что позволяет удалённо и одновременно нескольким пользователям работать с данными в таблицах.

4 Реализация программного средства

4.1 Реализация классов для работы с данными

Связь ПС с базой данных реализована при помощи Entity Framework, с применением подхода Code First. Его суть – сначала пишется код модели на C#, а затем по нему генерируется база данных [5]. Было разработано 4 модели User, Person, Assessment и Internship. Так же был разработан класс DataBaseContext, выполняющий роль входной точки для работы с БД, который содержит репозитории типа DbSet ранее упомянутых моделей. Подробное содержание выше приведённых классов находится в графических материалах, код реализации в приложении А.

В моделях используются атрибуты, для наложения ограничений целостности.

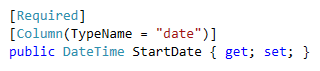


Рисунок 4.1 – Пример реализации поля StartDate

Атрибут Required уведомляет Entity Framework о том, что созданный по коду столбец должен иметь ограничение целостности NOT NULL.

Второй атрибут Column позволяет задать конкретный тип, в котором разработчик хочет хранить это поле в базе данных. В реализации полей для дат применены такие типы как: date – только дата, datetime – дата и время. Тип datetime используется в классе Assessment.

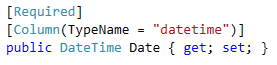


Рисунок 4.2 – Поле Date класса Assessment

Для инкапсуляции создания контекста данных от реализации интерфейса ПС, так же были введены классы сервисов, представляющие репозитории. Данные классы выполняют такую функцию как: создание контекста, работа с сущностями Entity Framework.

Методы данных классов реализуют асинхронный доступ к данным, и их возвращаемым значением в основном является тип Task.

Так же используется технология Linq to Entities, позволяющая работать в привычном стиле с сущностями фреймворка.

На примере класса UsersService разберём основные приёмы при работе с Entity Framework.

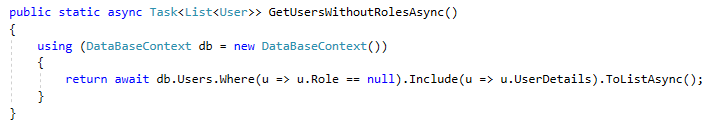


Рисунок 4.3 – Реализация метода на получение пользователей

На рисунке 4.3 показана реализация метода GetUsersWithoutRolesAsync, который делает выборку не подтверждённых пользователей.

В данном методе используются такие методы раширения как Where, который позволяет выбрать строки по задонному параметру, Include, позволяющий осуществить «жадную» загрузку вложенных объектов и ToListAsync который асинхронно создаёт список объектов.

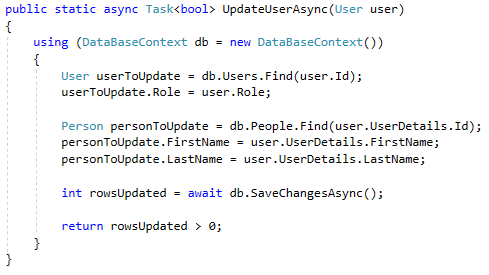


Рисунок 4.4 – Реализация метода на изменение пользователя

На рисунке 4.4 показана реализация метода UpdateUserAsync, который изменяет пользователя.

Для согласования объектов, сначала из текущего контекста получаем заданного пользователя и его детали, с помощью метода Find, который принимает идентификатор как параметр. Далее заменяем у полученных сущностей поля, требующие обновления и в конце вызываем метод контекста SaveChangesAsync, асинхронно сохраняющий наши изменения в БД. Возвращаемым значением является Task<bool>, чтобы получить подтверждение об изменении БД.

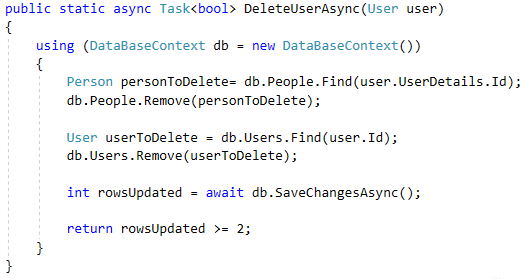


Рисунок 4.5 – Реализация метода на удаление пользователя

На рисунке 4.5 показана реализация метода DeleteUserAsync, который удаляет пользователя.

В данном методе используется такой метод как Remove, удаляющий заданную сущность из базы данных. В конце так же вызывается метод SaveChangesAsync.

Так же рассматриваемый класс содержит такие методы как: CheckIfUserExistsByEmailAsync – проверяет существования пользователя по email, GetUserByPersonIdAsync – возвращает пользователя по идентификатору, GetUserByEmailAsync – возвращает пользователя по email. Данные методы реализованны аналогично выше рассмотренным, с использованием методов расширения SingleOrDefaultAsync и AnyAsync.

Далее будет кратко рассмотрена реализация классов других сервисов, т.к. подход является аналогичным. Подробная реализация всех сервисов находится в приложении Б.

PeopleService содержит два метода: UpdatePersonAsync – обновление сущностей Person, UpdatePersonImageAsync – обновление конкретнго поля Image.

InternshipsService содержит методы: GetInternshipsByManagerIdAsync – получить интернатуры менеджера, AddInternshipAsync – добавить интернатуру, GetInternshipByInternIdAsync – получить интернатуру по Id интерна, CompleteInternshipAsync – завершить стажировку, UpdateInternshipEndDateAsync – обновить дату окончания стажировки.

AssessmentsService содержит методы: GetAssessmentsByInternIdAsync – получить встречи стажировки по Id интерна, AddAssessmentAsync – добавить встречу, DeleteAssessmentAsync – удалить встречу.

Отдельным классов является MailsService, т.к. он работает не с базой данных, а с SMTP сервером. Данный класс реализует один метод: SendEmailAsync.



Рисунок 4.6 – Реализация метода на отправку email

Метод SendEmailAsync принимает 4 параметра: адрес получателя emailTo, заголовок title, имя отправителя displayName, тело сообщения htmlBody.

Для обеспечения секретности пароля, используется класс AesEncryption, который будет рассмотрен далее. Так же данные авторизации хранятся в конфигурационном файле приложения, пароль зашифрован.

Далее с помощью встроенных классов формируется сообщение и отправляется через сервер smtp.gmail.com, 587 порт.

4.2 Реализация классов шифрования

Для шифрования пароля от smtp сервера был введён класс AesEncryption, подробная реализация находится в приложении В. Данный класс использует встроенный класс System.Security.Cryptography.Aes. AES – симметричный итеративный блоковый алгоритм шифрования [6]. Т.к. алгоритм симметричный, то ключ для шифровки и расшифровки одинаковый.

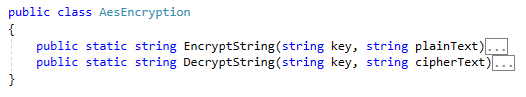


Рисунок 4.7 – Класс AesEncryption

Класс AesEncryption содержит в себе два метода: EncryptString – зашифровать строку по ключу и DecryptString – расшифровать строку по ключу.

Следующим классом шифрования является PasswordEncoder. Он предназначен для хеширования и проверки пароля пользователя.

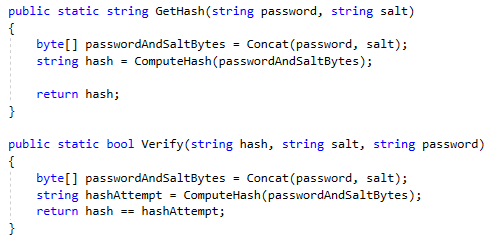


Рисунок 4.8 – Публичные методы PasswordEncoder

В методе GetHash с помощью приватных методов сначала получают массив байт, полученный из пароля и «соли». Соль нужна для предотвращения полной утечки паролей пользователей, без неё злоумышленник с помощью общедоступных утилит может распознать алгоритм хеширования и получить доступ сразу ко вмем паролям. При использовании динамической соли злоумышленнику придётся расшифровывать каждый пароль по отдельности.

Далее используется функция ComputeHash которая реализованна по алгоритму SHA256.

Метод Verify повторяет логику GetHash и сравнивает хеши паролей.

Подробную реализацию вышеприведённых классов можно найти в приложении В.

4.3 Реализация вспомогательных классов

Главным вспомогательным классом является AppNavHelper, реализованный по шаблону Singleton. Данный класс инициализируется в классе MainWindowView. Члены класса приведены на рисунке 4.9.

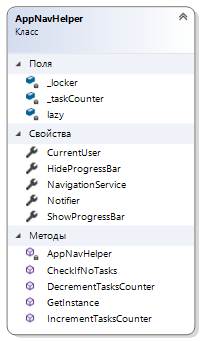


Рисунок 4.9 – Члены класса AppNavHelper

Свойство CurrentUser хранит текущего пользователя.

Свойства HideProgressBar и ShowProgressBar являются делегатами типа Action. Они нужны для управления видимостью бара загрузки.

Свойство NavigationService хранит одноименную сущность класса MainWindowView.

Свойство Notifier хранит в себе сущность для осуществления нотификации.

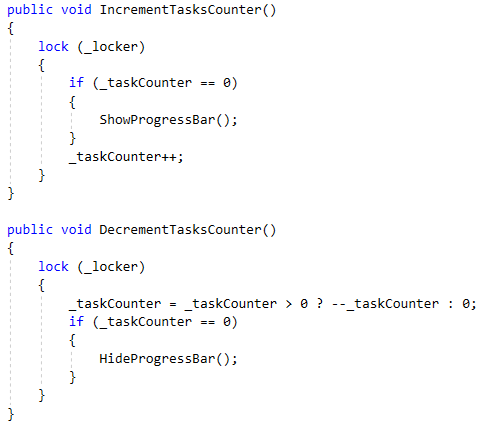


Рисунок 4.10 – Методы класса AppNavHelper

Т.к. приложение работает асинхронно, данный класс имеет механизмы синхронизации, такие как lazy инициализация самого класса и имплементацию критических секций в методах IncrementTasksCounter и DecrementTasksCounter. В упомянутых методах находится логика отображения бара загрузки.

Метод CheckIfNoTasks нужен для блокировки функционала интерфейса во время загрузки данных.

Следующим важным классом является ImageEncodingHelper, который осуществляет вызов диалога для выбора файла и преобразует png картинку в массив байтов.

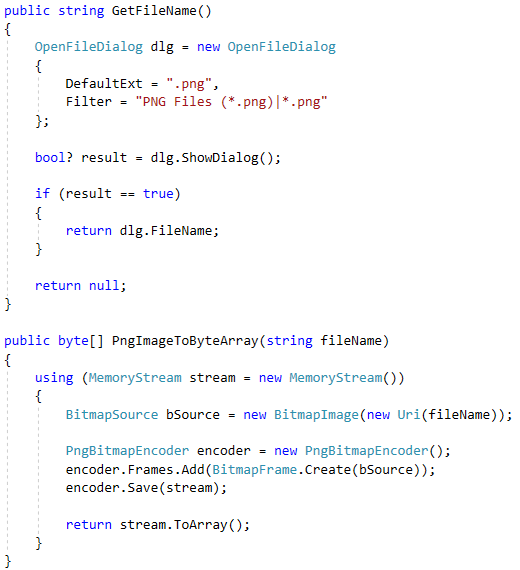


Рисунок 4.11 – Методы класса ImageEncodingHelper

Метод GetFileName реализован с помощью класса OpenFileDialog из пространства имён Microsoft.Win32. При вызове применены фильтры для типов файлов, в данном случае только png.

Метод PngImageToByteArray с помощью класса PngBitmapEncoder преобразует картинку в поток байт, а затем в массив.

4.4 Реализация классов интерфейса

Начнём с классов валидации. Для целей КП потребовалось две прверки: на email и на не пустое поле. Были введены два класса, наследуемые от ValidationRule: EmailRule и NotEmptyRule. Данные классы можно применять как правила валидации в коде xaml.

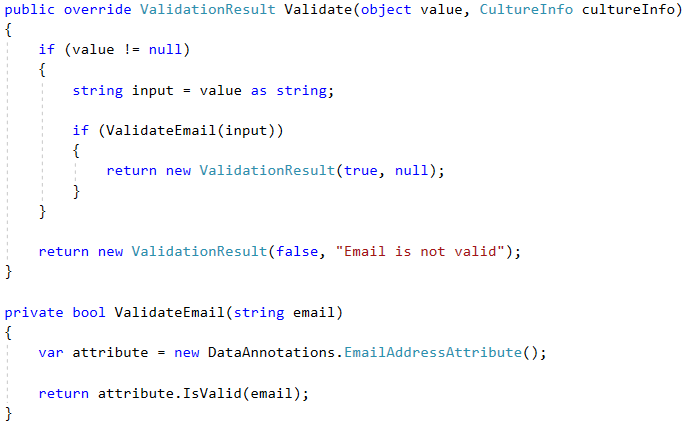


Рисунок 4.12 – Методы класса EmailRule

Суть валидации заключается в создании EmailAddressAttribute, который имеет метод IsValid и непосредственной проверке входной строки через этот атрибут.

Класс NotEmptyRule имеет анологичный вид, но в последнем условном блоке проверяется длина строки.

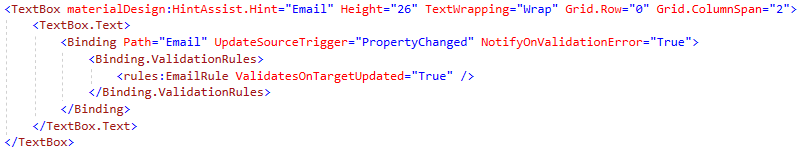


Рисунок 4.13 – Пример использования

Вышеприведённые классы можно использовать через привязку правил валидации.

Далее рассмотрим абстрактный класс PageWithValidation, необходимый для обработки ошибок, полученных от правил валидации.

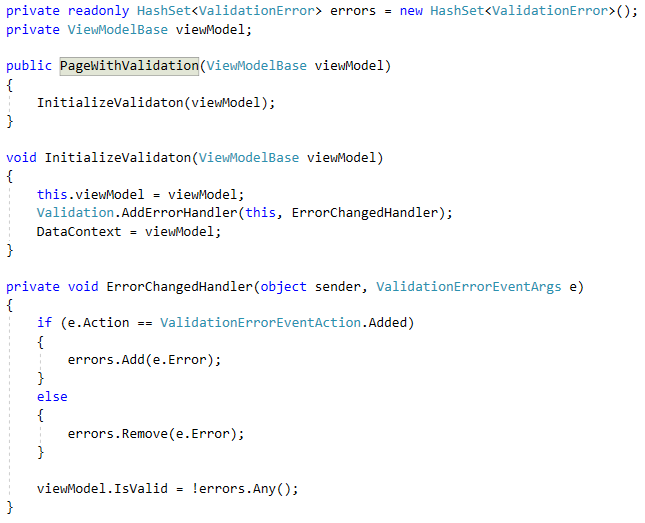


Рисунок 4.14 – Класс PageWithValidation

Данный класс агрегирует все ошибки ValidationError в коллекцию HashSet. Далее если коллекция не пуста, страница не считается валидной.

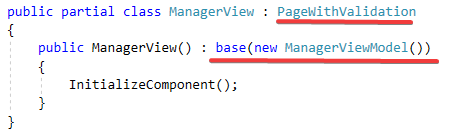


Рисунок 4.15 – Пример использования PageWithValidation

Далее переходим к рассмотрению классов моделей. Кроме ранее рассмотренных моделей Enitity Framework так же используются следуюшие модели:

Модель NewAssessment, состоящая из полей:

* Дата без времени Date;
* Время Time;
* Место Location;
* Тема Topic.

Модель PendingUser, состоящая из полей:

* Идентификатор Id;
* Идентификатор соответствующего человека DetailsId;
* Email;
* Тема Topic.
* Имя FirstName;
* Фамилия LastName.

Модель UserCredentials, состоящая из полей:

* Email;
* Пароль Password.

Модель UserSignUpCredentials, состоящая из полей:

* Email;
* Пароль Password;
* Имя FirstName;
* Фамилия LastName.

Следующая важная часть паттерна MVVM – модель представления.

Все классы ViewModel наследуются от абстрактного класса ViewModelBase, который реализует интерфейс INotifyPropertyChanged.

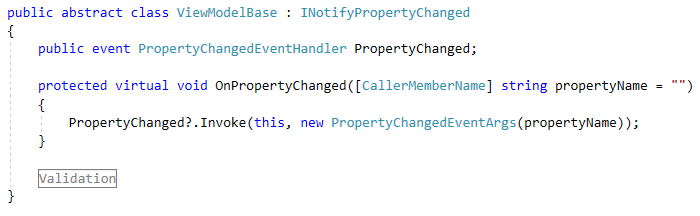


Рисунок 4.16 – Класс ViewModelBase

Интерфейс INotifyPropertyChanged реализован в абстрактном классе, т.к. он нужен всем ViewModel для получения уведомлений о изменении того или иного привязанного свойства.

Важной частью паттерна MVVM является реализация паттерна Command, т.к. представление не должно быть связанно с конкретным кодом. Рассмотрим реализацию классов Command и AsyncCommand.

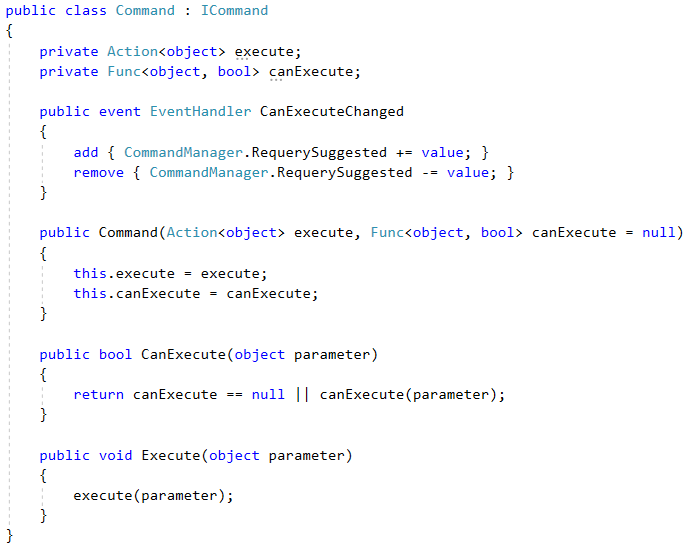


Рисунок 4.16 – Класс Command

Реализация класса AsyncCommand приведена в приложении Г.

Главное отличие данных классов заключается в том, что AsyncCommand имеет дополнительный метод ExecuteAsync, который помогает инкапсулировать асинхронное выполнение команды. Так же AsyncCommand имеет доступ к классу AppNavHelper, для обновления счётчика задач.

Обе команды состоят из двух основных методов: Execute – непосредственное выполнение и CanExecute – предикат, значение которого влияет на то, будет ли разрешено выполнение команды. Так же многие элементы управления имеют встроенное поведение на изменение значения CanExecute, к примеру элемент Button меняет состояние активности.

После рассмотрения основных составляющих ViewModel, перейдём к их реализации. В качестве примера рассмотрим AuthorizationViewModel.



Рисунок 4.17 – Поля и свойства класса AuthorizationViewModel

В начале происходит инициализация \_appNavHelper и модели UserCredentials. Далее происходит выделение полей модели в свойства модели представления и вызов функции для уведомления об изменении свойства. Последним идёт свойство ErrorMessage, необходимое для вывода ошибок авторизации.



Рисунок 4.18 – Пример комманды класса AuthorizationViewModel

В данном примере отражена асинхронная команда входа в систему. Чтобы не блокировать поток UI, проверка пользователя в БД выносится в отдельную задачу, что позволяет запустить данную операцию в другом потоке. Здесь используется ранее рассмотренная функция Verify, в параметрах которой соль является email’ом пользователя. Так же в параметры конструктора команды передана анонимная функция CanExecute, которая не даёт выполнить команду если есть ошибки валидации или идёт загрузка данных.

Ещё AuthorizationViewModel содержит команду SignUpCommand, для перехода на страницу регистрации.

Остальные ViewModel будут рассмотрены кратко, но ещё к рассмотрению необходим пример загрузки данных для отображения, для этих нужд следующим возьмём InternViewModel.

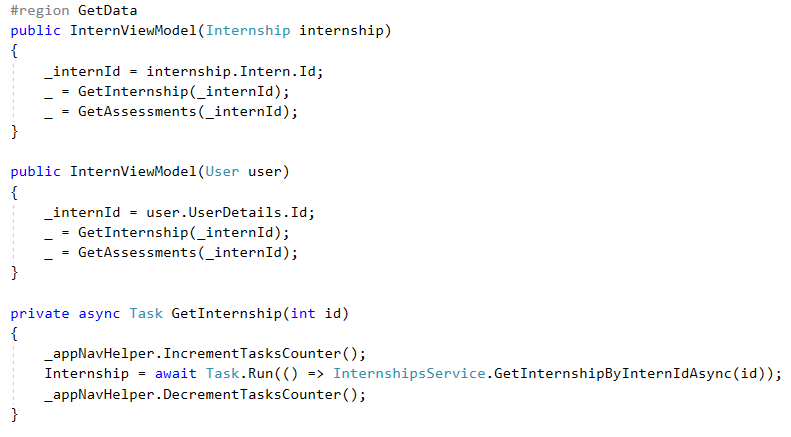


Рисунок 4.19 – Конструкторы InternViewModel

Модель представления имеет два конструктора, т.к. в зависимости от роли пользователя, входные параметры будут различаться. В конструкторе мы асинхронно вызываем методы загрузки данных и их возвращаемое значение присваиваем пустой переменной, т.к. результат операции в конструкторе нам не нужен. Так же функции загрузки данных могут быть вызваны в любой точке кода.

Остальные члены InternViewModel и других моделей представления находятся на диаграмме классов в графических материалах.

Далее рассмотрим последнюю часть паттерна MVVM – представления. При использовании паттерна MVVM представление никак не зависит от других частей программы и её ViewModel может быть легко заменена на другую. В .cs файле находится только инициализация представления и его привязка к ViewModel. Главная часть представления – это xaml код. Приведём несколько основных примеров xaml разметки:

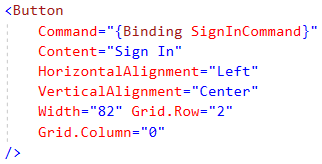


Рисунок 4.20 – Пример реализации кнопки

В данном примере видно применение стилей и привязку команды к кнопке.



Рисунок 4.21 – XAML код MainViewWindow

Здесь мы прописываем заголовок окна программы, иконку, фон и стиль шрифта. Так же данном классе мы используем ЭУ Frame, чтобы сделать страничную навигацию, размещаем бар загрузки снизу страницы.

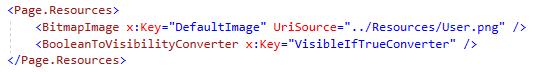


Рисунок 4.22 – Пример добавления ресурсов страницы

В данном примере мы сохраняем в ресурсы временную картинку пользователя и конвертер BooleanToVisibilityConverter, позволяющий привести значение свойств типа bool к типу Visibility. Ниже рассмотрим использование этих ресурсов.

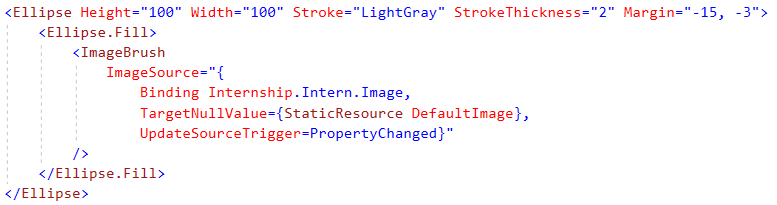


Рисунок 4.23 – Пример реализации аватара

В приведённой реализации аватара используется элемент Ellipse, и он заполняется картинкой аватара. Если картинка пользователя не была загружена, то используется картинка по умолчанию из ресурса страницы.

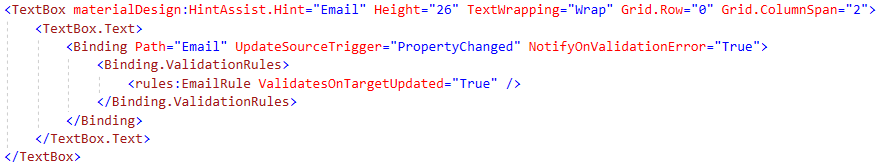


Рисунок 4.24 – Текстовое поле с валидацией

В текстовых полях используется вспомогательный атрибут Hint из библиотеки Material Design, позволяющий добавить текст подсказки к ЭУ. Так же в примере реализована валидация, через вышеупомянутые правила валидации.



Рисунок 4.25 – ResourceDictionary приложения

На картинке отображены подключенные к приложению словари ресурсов, для работы пакетов Material Design и Toast Notifications.

4.5 Сборка проекта

Сборка для разработки происходит автоматически и не требует дополнительных настроек.

Для создания установочного файла было установлено расширение Microsoft Visual Studio Installer Projects и создан проект типа Setup Project.

Сборка проекта в установочный файл происходит в следующем порядке:

* Открыть вкладку «Сборка» в Visual Studio.
* Выбрать пункт «Пакетная сборка»;
* Выбрать для сборки Release версии проектов «WpfClient» и «Installer»;
* Нажать на кнопку «Сборка» (либо «Перестроить»).

5 Тестирование, проверка работоспособности и анализ полученных результатов

5.1 Тестирование страницы авторизации

Авторизация в обычной ситуации работает стабильно.

Может возникнуть ситуация, когда пользователь отключен от сети или база данных не доступна, проверим поведение системы в данном случае.

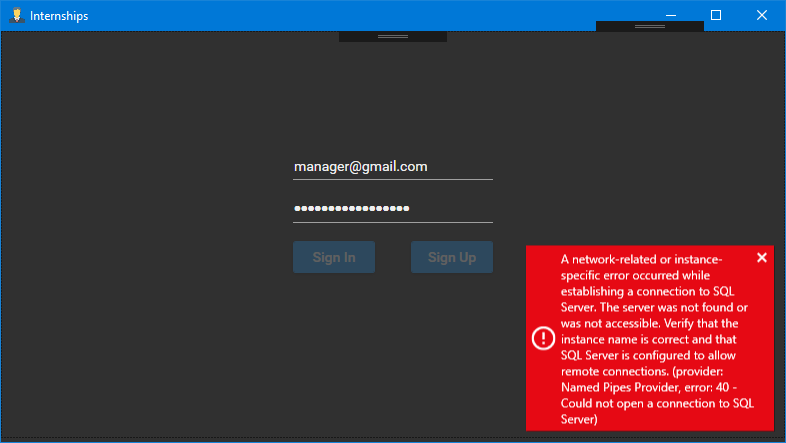


Рисунок 5.1 – Результат теста при отключенной БД

Как видим критическая ситуация обработанна.

Далее перейдём к валидации полей.

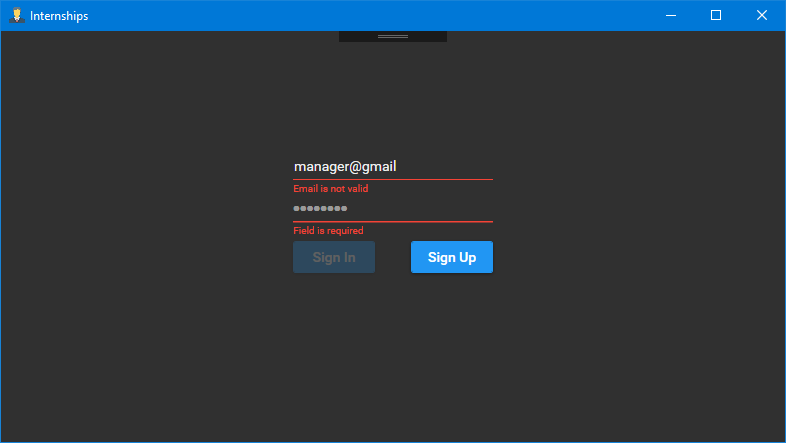


Рисунок 5.2 – Результат теста при неверном вводе данных

На представленном рисунке видно, что email и пароль не соответсвуют правилам валидации и по этому кнопка отключена.

Далее рассмотрим валидацию полей, с подключением к БД.

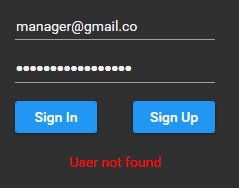


Рисунок 5.3 – Результат теста при несуществующем email

На представленном рисунке видно, что пользователь по заданному email не найден.

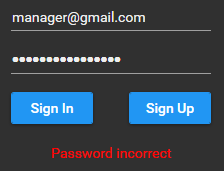


Рисунок 5.4 – Результат теста при неверном пароле

На представленном рисунке видно, что пароль введён не верно.

5.2 Тестирование страницы регистрации

Регистрация в обычной ситуации работает стабильно.

Рассмотрим валидацию полей.

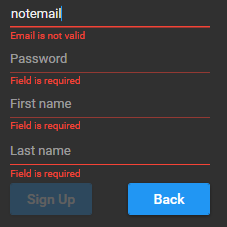


Рисунок 5.5 – Результат теста при неверных данных

Далее рассмотрим валидацию полей, с подключением к БД.

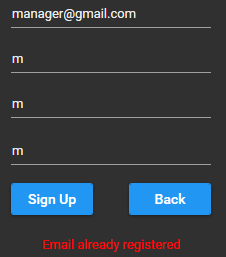


Рисунок 5.6 – Результат теста при уже зарегистрированном email

5.3 Тестирование страницы не проверенных пользователей

Попытаемся зайти с только что зарегистрированного аккаунта.

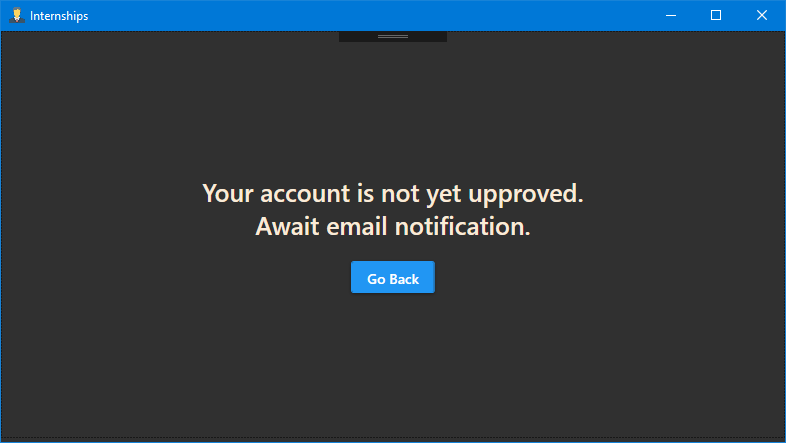


Рисунок 5.7 – Страница не подтверждённого пользователя

Получили отказ, т.к. текущему пользователю не выдана роль. На странице менеджера можно будет выдать роль пользователю: интерн или менеджер. Так же возможно удаление аккаунта.

5.4 Тестирование страницы менеджера

Авторизуемся под аккаунтом менеджера.

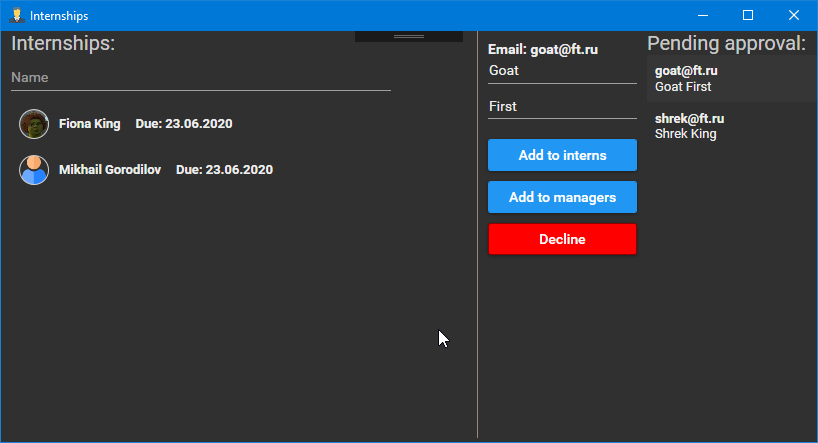


Рисунок 5.8 – Страница менеджера

Видим стартовую страницу, слева список интернов, справа пользователей которым нужно выдать роль.

Функциональность списка ждущих пользователей реализована в полной мере.

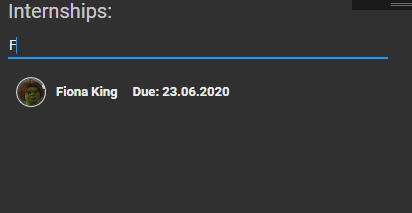


Рисунок 5.9 – Тестирование поиска

Переход к конкретному интерну из списка также работает.

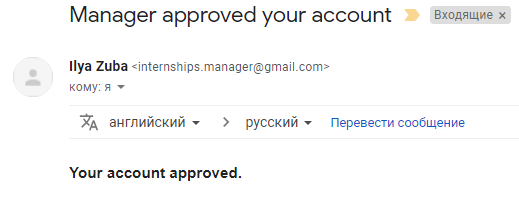


Рисунок 5.10 – Проверка email нотификации

Email нотификация о подтверждении аккаунта получена.

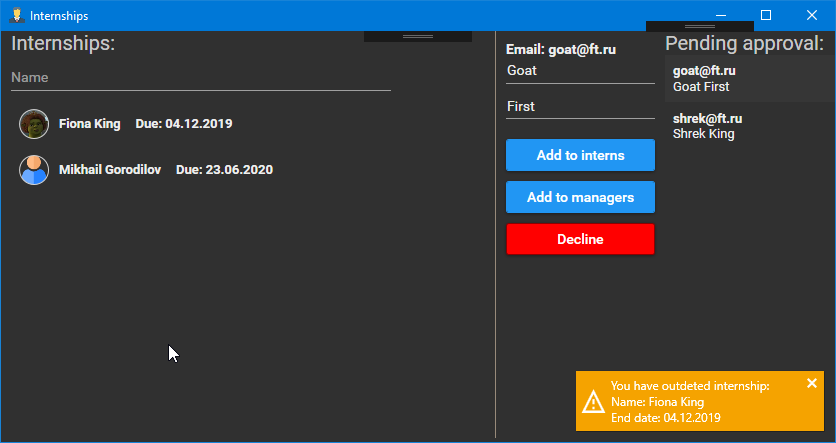


Рисунок 5.10 – Проверка нотификации о просроченной стажировке

Если присутствует просроченная стажировка, появляется всплывающее уведомление.

5.5 Тестирование страницы интерна

Сначала зайдём на страницу под аккаунтом менеджера.

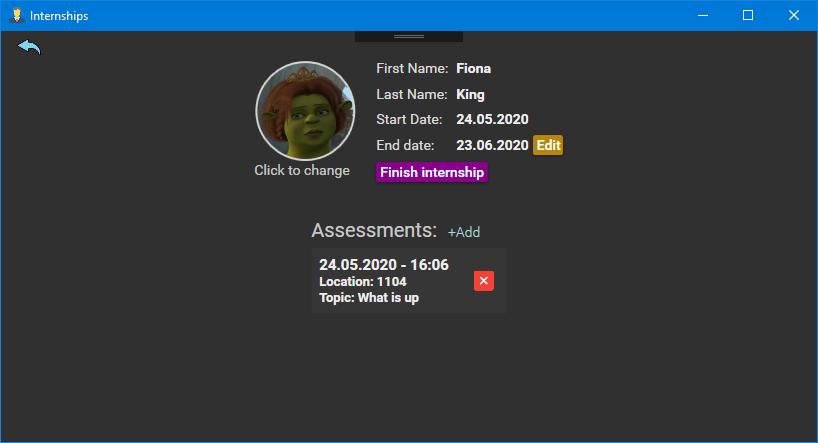


Рисунок 5.11 – Страница интерна

На данной странице мы можем поменять дату окончания, назначать и удалять встречи (все даты только с начала стажировки), закончить стажировку, а так же менять аватар. Ниже приведена изменённая страница.

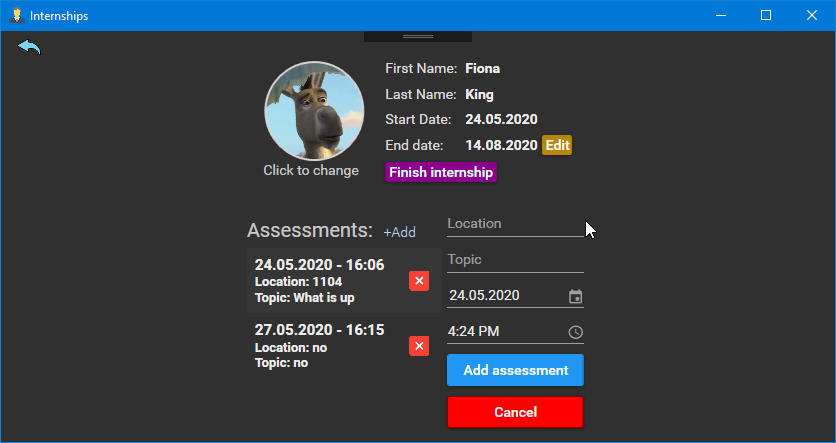


Рисунок 5.12 – Изменённая страница интерна

При завершении стажировки пользователя перенаправляет обратно к списку интернов.

При авторизации через интерна мы увидим следующую страницу.

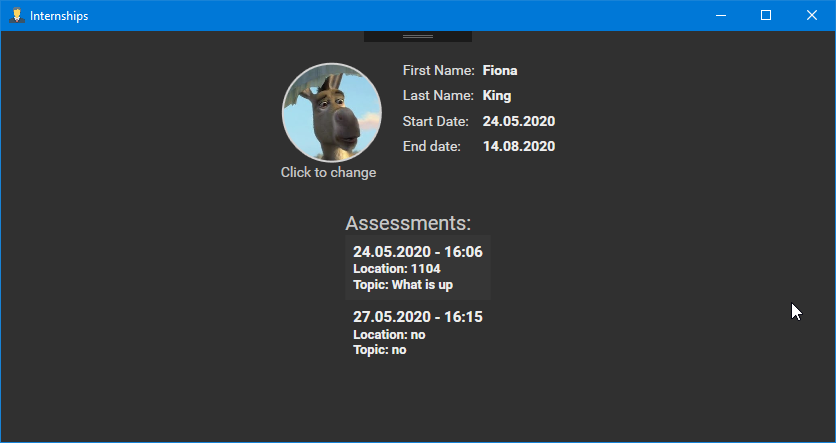


Рисунок 5.13 – Страница интерна без прав редактирования

Далее приведены примеры email нотификации при добавлении и удалении встречи.

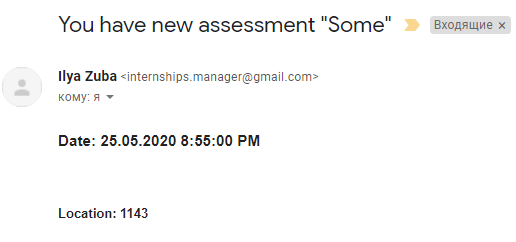


Рисунок 5.14 – Уведомление о встрече

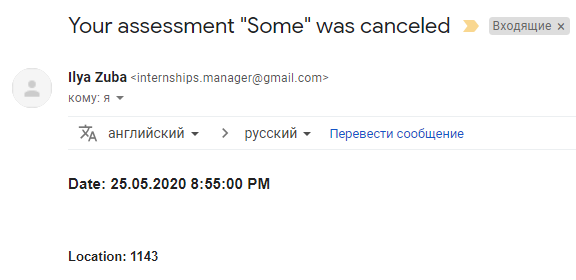


Рисунок 5.15 – Уведомление об отмене встречи

Ниже приведён пример email при окончании стажировки, письмо так же включает в себя её продолжительность.

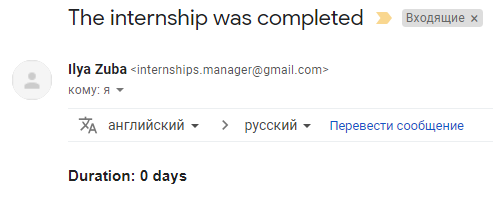


Рисунок 5.16 – Уведомление об окончании

6 Руководство по установке и использованию

6.1 Системные требования

В первую очередь используемое устройство должно быть подключено к сети интернет (для полного функционирования) и находится в локальной сети с развёрнутым SQL сервером (для минимально необходимого функционирования). Так же должен быть установлен .NET Framework.

Поддерживаемые операционные системы: Windows 10, Windows 8, Windows 7, большинство версий Windows Server.

Поддерживаемая разрядность: x86 и x64.

Минимальные требования к оборудованию, обусловленные .NET Framework:

* Частота процессора 1 ГГц и более;
* Оперативная память 512 МБ;
* Место на диске 4.5 ГБ.

6.2 Установка

Запустите полученный файл с расширением msi.



Рисунок 6.1 – Установочный файл

Выберите нужный вам каталог установки (если требуется).

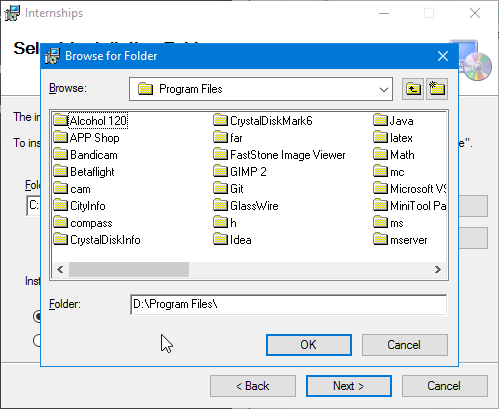


Рисунок 6.2 – Выбор каталога

Далее нужно согласится на установку. После подтверждения программа будет установлена и на рабочем столе появится её ярлык.

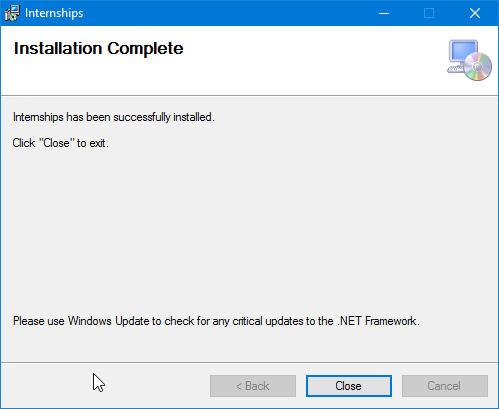


Рисунок 6.3 – Завершение установки

Далее программу можно запускать с рабочего стола.

Предыдущий раздел полностью повторяет руководство пользователя, по этому при необходимости обращайтесь к нему.

ЗАКЛЮЧЕНИЕ

В процессе курсового проектирования было разработанно программное средство для управления интернами. Были выполнены все требования к ПС. Таким образом цель курсового проекта была достигнута.

В программном средстве был реализован следующий функционал:

* Возможность авторизации и регистрации;
* Интерфейс для управления заявками на создание аккаунта;
* Возможность просматривать список стажировок и поиск по нему;
* Возможность менять картинку профиля;
* Возможность выставления временных рамок стажировки;
* Интерфейс для частичного редактирования стажировки;
* Возможность назначать встречу со студентом;
* Возможность завершения стажировки;
* Оповещения в программе;
* Оповещения через email;

В переспективе в систему нужно добавить клиент-серверную архитектуру, чтобы достичь большей сохранности данных и иметь возможность реализовать клиентские приложения под разные операционные системы, а так же браузер.

Список использованных источников

1. Сайт обзоров на приложения: обзор на Connecteam [Электронный ресурс] / Режим доступа: http://www.saasgenius.com/program/instant-messaging-chat-software/connecteam Дата доступа: 10.03.2020

2. Сайт о программировании: паттерн MVVM [Электронный ресурс] / Режим доступа: https://metanit.com/sharp/wpf/22.1.php Дата доступа: 11.04.2020

3. Сайт о программировании: паттерн Command [Электронный ресурс] / Режим доступа: https://refactoring.guru/ru/design-patterns/command Дата доступа: 19.04.2020

4. Сайт документации Microsoft: свойство IDENTITY [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql-identity-property?view=sql-server-ver15 Дата доступа: 25.04.2020

5. Сайт базы знаний: Entity Framework [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/ADO.NET\_Entity\_Framework Дата доступа: 29.04.2020

6. Общее описание алгоритма AES [Электронный ресурс] / Режим доступа: http://bit.nmu.org.ua/ua/student/metod/cryptology/%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%209.pdf Дата доступа: 3.05.2020

ПРИЛОЖЕНИЕ А

Листинг 1. Код контекста Enitity Framework.

using System.Data.Entity;

using WpfClient.DataBase.Models;

namespace WpfClient.DataBase

{

public class DataBaseContext : DbContext

{

public DataBaseContext()

{ }

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

public DbSet<Person> People { get; set; }

public DbSet<Internship> Internships { get; set; }

public DbSet<Assessment> Assessments { get; set; }

}

}

Листинг 2. Код моделей Enitity Framework.

public class Assessment

{

public int Id { get; set; }

[Required]

public Internship Internship { get; set; }

[Required]

[Column(TypeName = "datetime")]

public DateTime Date { get; set; }

public string Location { get; set; }

public string Topic { get; set; }

}

public class Internship

{

public int Id { get; set; }

[Required]

public Person Intern { get; set; }

public virtual Person Manager { get; set; }

[Required]

[Column(TypeName = "date")]

public DateTime StartDate { get; set; }

[Required]

[Column(TypeName = "date")]

public DateTime EndDate { get; set; }

public bool? IsCompleted { get; set; }

}

public class Person

{

public int Id { get; set; }

[Required]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

public virtual byte[] Image { get; set; }

}

public class User

{

public int Id { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string HashedPassword { get; set; }

[Required]

public Person UserDetails { get; set; }

public Role? Role { get; set; }

}

public enum Role { Manager, Intern }

ПРИЛОЖЕНИЕ Б

Листинг 1. Код сервиса UsersService.

public class UsersService

{

public static async Task<User> GetUserByEmailAsync(string email)

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Users.Where(u => u.Email == email).Include(u => u.UserDetails).SingleOrDefaultAsync();

}

}

public static async Task<User> GetUserByPersonIdAsync(int personId)

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Users.Where(u => u.UserDetails.Id == personId).SingleOrDefaultAsync();

}

}

public static async Task<bool> AddUserAsync(User user)

{

using (DataBaseContext db = new DataBaseContext())

{

db.Users.Add(user);

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<bool> CheckIfUserExistsByEmailAsync(string email)

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Users.AnyAsync(u => u.Email == email);

}

}

public static async Task<List<User>> GetUsersWithoutRolesAsync()

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Users.Where(u => u.Role == null).Include(u => u.UserDetails).ToListAsync();

}

}

public static async Task<bool> UpdateUserAsync(User user)

{

using (DataBaseContext db = new DataBaseContext())

{

User userToUpdate = db.Users.Find(user.Id);

userToUpdate.Role = user.Role;

Person personToUpdate = db.People.Find(user.UserDetails.Id);

personToUpdate.FirstName = user.UserDetails.FirstName;

personToUpdate.LastName = user.UserDetails.LastName;

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<bool> DeleteUserAsync(User user)

{

using (DataBaseContext db = new DataBaseContext())

{

Person personToDelete= db.People.Find(user.UserDetails.Id);

db.People.Remove(personToDelete);

User userToDelete = db.Users.Find(user.Id);

db.Users.Remove(userToDelete);

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated >= 2;

}

}

}

Листинг 2. Код сервиса PeopleService.

public class PeopleService

{

public static async Task<bool> UpdatePersonAsync(Person person)

{

using (DataBaseContext db = new DataBaseContext())

{

Person personToUpdate = db.People.Find(person.Id);

personToUpdate.FirstName = person.FirstName;

personToUpdate.LastName = person.LastName;

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<bool> UpdatePersonImageAsync(int personId, byte[] img)

{

using (DataBaseContext db = new DataBaseContext())

{

Person personToUpdate = db.People.Find(personId);

personToUpdate.Image = img;

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

}

Листинг 3. Код сервиса InternshipsService.

public class InternshipsService

{

public static async Task<List<Internship>> GetInternshipsByManagerIdAsync(int managerId, bool includeFinished = false)

{

using (DataBaseContext db = new DataBaseContext())

{

var query = db.Internships.Where(i => i.Manager.Id == managerId);

if (!includeFinished)

{

query = query.Where(i => i.IsCompleted != true);

}

return await query.Include(i => i.Intern).ToListAsync();

}

}

public static async Task<bool> AddInternshipAsync(Internship internship)

{

using (DataBaseContext db = new DataBaseContext())

{

internship.Intern = db.People.Find(internship.Intern.Id);

internship.Manager = db.People.Find(internship.Manager.Id);

db.Internships.Add(internship);

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<Internship> GetInternshipByInternIdAsync(int internId)

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Internships.Where(i => i.Intern.Id == internId).Include(i => i.Intern).SingleOrDefaultAsync();

}

}

public static async Task<bool> CompleteInternshipAsync(int internshipId)

{

using (DataBaseContext db = new DataBaseContext())

{

Internship internship = await db.Internships.FindAsync(internshipId);

internship.IsCompleted = true;

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<bool> UpdateInternshipEndDateAsync(Internship internship)

{

using (DataBaseContext db = new DataBaseContext())

{

Internship internshipToUpdate = await db.Internships.FindAsync(internship.Id);

internshipToUpdate.EndDate = internship.EndDate;

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

}

Листинг 4. Код сервиса AssessmentsService.

public class AssessmentsService

{

public static async Task<List<Assessment>> GetAssessmentsByInternIdAsync(int internId)

{

using (DataBaseContext db = new DataBaseContext())

{

return await db.Assessments.Where(a => a.Internship.Intern.Id == internId).ToListAsync();

}

}

public static async Task<bool> AddAssessmentAsync(Assessment assessment)

{

using (DataBaseContext db = new DataBaseContext())

{

Internship internship = await db.Internships.FindAsync(assessment.Internship.Id);

assessment.Internship = internship;

db.Assessments.Add(assessment);

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

public static async Task<bool> DeleteAssessmentAsync(int assessmentId)

{

using (DataBaseContext db = new DataBaseContext())

{

Assessment assessment = await db.Assessments.FindAsync(assessmentId);

db.Assessments.Remove(assessment);

int rowsUpdated = await db.SaveChangesAsync();

return rowsUpdated > 0;

}

}

}

Листинг 5. Код сервиса MailsService.

public class MailsService

{

public static async Task SendEmailAsync(string emailTo, string title, string displayName, string htmlBody = "")

{

string login = ConfigurationManager.AppSettings["emailLogin"];

string password = ConfigurationManager.AppSettings["emailPass"];

password = AesEncryption.DecryptString("b14ca5898a4e4133bbce2ea2315a1916", password);

MailAddress from = new MailAddress(login, displayName);

MailAddress to = new MailAddress(emailTo);

MailMessage message = new MailMessage(from, to)

{

Subject = title,

Body = htmlBody,

IsBodyHtml = true

};

SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587)

{

Credentials = new NetworkCredential(login, password),

EnableSsl = true,

};

await smtp.SendMailAsync(message);

}

}

ПРИЛОЖЕНИЕ В

Листинг 1. Код класса AesEncryption.

public class AesEncryption

{

public static string EncryptString(string key, string plainText)

{

byte[] iv = new byte[16];

byte[] array;

using (Aes aes = Aes.Create())

{

aes.Key = Encoding.UTF8.GetBytes(key);

aes.IV = iv;

ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV);

using (MemoryStream memoryStream = new MemoryStream())

{

using (CryptoStream cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))

{

using (StreamWriter streamWriter = new StreamWriter(cryptoStream))

{

streamWriter.Write(plainText);

}

array = memoryStream.ToArray();

}

}

}

return Convert.ToBase64String(array);

}

public static string DecryptString(string key, string cipherText)

{

byte[] iv = new byte[16];

byte[] buffer = Convert.FromBase64String(cipherText);

using (Aes aes = Aes.Create())

{

aes.Key = Encoding.UTF8.GetBytes(key);

aes.IV = iv;

ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

using (MemoryStream memoryStream = new MemoryStream(buffer))

{

using (CryptoStream cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))

{

using (StreamReader streamReader = new StreamReader(cryptoStream))

{

return streamReader.ReadToEnd();

}

}

}

}

}

}

Листинг 2. Код класса PasswordEncoder.

public static class PasswordEncoder

{

public static string GetHash(string password, string salt)

{

byte[] passwordAndSaltBytes = Concat(password, salt);

string hash = ComputeHash(passwordAndSaltBytes);

return hash;

}

public static bool Verify(string hash, string salt, string password)

{

byte[] passwordAndSaltBytes = Concat(password, salt);

string hashAttempt = ComputeHash(passwordAndSaltBytes);

return hash == hashAttempt;

}

private static string ComputeHash(byte[] bytes)

{

using (SHA256 sha256 = SHA256.Create())

{

return Convert.ToBase64String(sha256.ComputeHash(bytes));

}

}

private static byte[] Concat(string password, string salt)

{

byte[] passwordBytes = Encoding.UTF8.GetBytes(password);

byte[] saltBytes = Encoding.UTF8.GetBytes(salt);

return passwordBytes.Concat(saltBytes).ToArray();

}

}

ПРИЛОЖЕНИЕ Г

Листинг 1. Класс AsyncCommand

public class AsyncCommand : IAsyncCommand

{

private AppNavHelper \_appNavHelper = AppNavHelper.GetInstance();

private readonly Func<object, Task> \_command;

private Func<object, bool> canExecute;

public AsyncCommand(Func<object, Task> command, Func<object, bool> canExecute = null)

{

\_command = command;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

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

}

public Task ExecuteAsync(object parameter)

{

return \_command(parameter);

}

public async void Execute(object parameter)

{

try

{

\_appNavHelper.IncrementTasksCounter();

await ExecuteAsync(parameter);

}

catch (Exception e)

{

\_appNavHelper.Notifier.ShowError(e.Message);

}

finally

{

\_appNavHelper.DecrementTasksCounter();

}

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

protected void RaiseCanExecuteChanged()

{

CommandManager.InvalidateRequerySuggested();

}

}