МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение высшего образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Специализация 1-40 05 01-03 Информационные системы и

технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к дипломному проекту на тему:**

«Мессенджер со сквозным шифрованием»

Дипломник Гращенко Александр Сергеевич

(Ф.И.О.)

Руководитель проекта преподаватель-стажёр Нистюк О. А.

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

Заведующий кафедрой к.т.н., доц. Смелов В. В.

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

Консультанты преподаватель-стажёр Нистюк О. А.

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

к.э.н, доц. Евлаш А.И.

Нормоконтролер ст. преп. Рыжанкова А.С.

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

Дипломный проект защищен с оценкой 1

Председатель ГЭК к.т.н., доц. Дюбков В. К.

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

Минск 20201

Реферат

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 00.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | Реферат | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 1 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

Пояснительная записка содержит 60 страниц, 38 рисунка, 10 таблиц, 14 литературных источников.

Asp.Net Core, Angular, signalr, ms sql, ENTITYFRAMEWORK, IIs express.

Целью работы является разработка мессенджера со сквозным шифрованием.

Пояснительная записка дипломного проекта состоит из реферата, оглавления, пяти глав, заключения и списка использованных источников.

В первом разделе описаны задачи, которые ставятся при разработке программного средства, описаны аналоги данного веб-приложения, выделены плюсы и минусы каждого из них, произведен обзор теоретического материала, а также произведен патентный поиск по теме данного дипломного проекта.

Во втором разделе описаны программные средства и технологии, которые выбраны для разработки веб-приложения. Выбор сделан в пользу наиболее распространенных технологий.

В третьем разделе приведено описание процесса разработки программного обеспечения веб-приложения. Приводятся листинги исходного кода и комментарии к ним, показаны блок схемы.

В четвертом разделе приведены результаты тестирования веб-приложения.

В пятом разделе приведено руководство пользователя.

В шестом разделе приводится расчет экономических параметров и себестоимости программного средства.

В заключении приведены результаты проделанной работы, приводятся соображения насчет использования данного программного средства.

# Содержание

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 00.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | Содержание | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 2 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

[Содержание 3](#_Toc41333764)

[Введение 5](#_Toc41333765)

[1 Обзор теоретического материала и аналогов 6](#_Toc41333766)

[1.1 Постановка задачи 6](#_Toc41333767)

[1.2 Актуальность задачи 6](#_Toc41333768)

[1.3 Описание сквозного шифрования 7](#_Toc41333769)

[1.4 Схема шифрования 7](#_Toc41333770)

[1.5 Возможные атаки 8](#_Toc41333771)

[1.6 Обзор аналогов 9](#_Toc41333772)

[1.6.1 Telegram 9](#_Toc41333773)

[1.6.2 Whatsapp 10](#_Toc41333774)

[1.7 Патентный поиск по теме дипломного проекта 12](#_Toc41333775)

[1.8 Выводы по разделу 15](#_Toc41333776)

[2 Проектирование веб-приложения 16](#_Toc41333777)

[2.1 Обзор средств разработки 16](#_Toc41333778)

[2.1.1 Платформа .NET Core 16](#_Toc41333779)

[2.1.2 Платформа ASP.NET Core 16](#_Toc41333780)

[2.1.3 Среда разработки 18](#_Toc41333781)

[2.1.4 Обоснование выбора языка программирования 18](#_Toc41333782)

[2.1.5 Программное обеспечение сервера 19](#_Toc41333783)

[2.1.6 База данных MS SQL Server 19](#_Toc41333784)

[2.1.7 Entity Framework 20](#_Toc41333785)

[2.1.8 Angular 21](#_Toc41333786)

[2.1.9 Язык TypeScript 21](#_Toc41333787)

[2.1.10 SignalR 22](#_Toc41333788)

[2.2 Архитектура программного продукта 22](#_Toc41333789)

[2.3 Выводы по разделу 24](#_Toc41333790)

[3 Разработка программного обеспечения веб-приложения 25](#_Toc41333791)

[3.1 Проектирование программного средства 25](#_Toc41333792)

[3.2 Файловая структура 27](#_Toc41333793)

[3.2.1 WebApi 27](#_Toc41333794)

[3.2.2 Angular app 28](#_Toc41333795)

[3.3 Структура базы данных 29](#_Toc41333796)

[3.4 Разработка контроллера ChatController 32](#_Toc41333797)

[3.5 Разработка приложения Angular 33](#_Toc41333798)

[3.5.1 Сервисы 33](#_Toc41333799)

[3.5.2 Компоненты 35](#_Toc41333800)

[3.5.3 Модели 39](#_Toc41333801)

[3.6 Websocket 39](#_Toc41333802)

[3.7 Вывод по разделу 40](#_Toc41333803)

[4 Тестирование приложения 42](#_Toc41333804)

[4.1 Отправка сообщения 42](#_Toc41333805)

[4.2 Импорт ключа 43](#_Toc41333806)

[4.3 Экспорт ключа 44](#_Toc41333807)

[4.4 Отправка файла 44](#_Toc41333808)

[4.5 Заявка в друзья 45](#_Toc41333809)

[4.6 Удаление пользователя 46](#_Toc41333810)

[4.7 Вывод по разделу 46](#_Toc41333811)

[5 Руководство пользователя 47](#_Toc41333812)

[5.1 Системные требования 47](#_Toc41333813)

[5.2 Авторизация пользователя 47](#_Toc41333814)

[5.3 Регистрация пользователя 48](#_Toc41333815)

[5.4 Добавление пользователя в друзья 49](#_Toc41333816)

[5.5 Поиск по сообщениям 50](#_Toc41333817)

[5.6 Вывод по разделу 50](#_Toc41333818)

[6 Экономическое обоснование цены программного средства 51](#_Toc41333819)

[6.1 Общая характеристика разрабатываемого программного средства 51](#_Toc41333820)

[6.2 Исходные данные и маркетинговый анализ 52](#_Toc41333821)

[6.3 Методика обоснования цены 53](#_Toc41333822)

[6.4 Объем программного средства 53](#_Toc41333823)

[6.5 Основная заработная плата 54](#_Toc41333824)

[6.6 Дополнительная заработная плата 55](#_Toc41333825)

[6.7 Отчисления в Фонд социальной защиты населения 55](#_Toc41333826)

[6.8 Расходы на материалы 55](#_Toc41333827)

[6.9 Расходы на оплату машинного времени 56](#_Toc41333828)

[6.10 Прочие прямые затраты 56](#_Toc41333829)

[6.11 Накладные расходы 56](#_Toc41333830)

[6.12 Сумма расходов на разработку программного средства 57](#_Toc41333831)

[6.13 Расходы на сопровождение и адаптацию 57](#_Toc41333832)

[6.14 Полная себестоимость 57](#_Toc41333833)

[6.15 Определение цены, оценка эффективности 58](#_Toc41333834)

[6.16 Вывод по разделу 58](#_Toc41333835)

[Заключение 60](#_Toc41333836)

[ПРИЛОЖЕНИЕ A 62](#_Toc41333837)

[ПРИЛОЖЕНИЕ Б 68](#_Toc41333838)

[ПРИЛОЖЕНИЕ В 71](#_Toc41333839)

[ПРИЛОЖЕНИЕ Г 76](#_Toc41333840)

[ПРИЛОЖЕНИЕ Д 78](#_Toc41333841)

[ПРИЛОЖЕНИЕ Е 81](#_Toc41333842)

[ПРИЛОЖЕНИЕ Ж 87](#_Toc41333843)

Введение

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 00.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | Введение | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 1 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

Мессенджер – средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение. Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает мессенджер от форумов и других «медленных» средств. Наиболее удобной формой чата является веб-мессенджер. Веб-мессенджер представляет собой веб-страницу в браузере для обмена сообщениями в режиме реального времени.

Кроме скорости и удобства интерфейса, важнейшим свойством чата является безопасность. Как правило, ключи шифрования хранятся на сервере, что делает безопасную передачу данных зависящей от надежности сервера. Многих пользователей не устраивает такой способ хранения их конфиденциальных данных. Чтобы обойти это, была придумана модель сквозного шифрования.

Сквозное шифрование гарантирует, что доступ к исходному тексту сообщения имеется только у отправителя и получателя. Это означает, что пользовательская информация становится недоступной даже серверам, передающим данные.

Шифрование и дешифрование происходит на конечных устройствах пользователей. Кроме того, данные остаются зашифрованными, пока не будут доставлены к месту назначения. Поэтому часто сквозное шифрование также называют «нулевой доступ» или «шифрование на стороне клиента». Однако, следует различать оконечное шифрование при передаче данных и шифрование на стороне клиента при хранении данных.

Исходя из этого было разработано веб-приложение «Мессенджер со сквозным шифрованием».

Также были поставлены такие задачи, как:

– проектирование и разработка структуры базы данных;

– разработка интерфейса веб-приложения;

– реализация алгоритма обмена ключевой информации;

– реализация импорта и экспорта ключевой информации;

– реализация загрузки сообщений по мере прокрутки страницы;

– проектирование формата хранения ключевой информации в браузере и в файлах для импорта;

– реализация авторизации и аутентификации на основе *ASP.NET CORE Identity*;

– разработка веб-сокета для отправки сообщений;

1. Обзор теоретического материала и аналогов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 01.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 1 Обзор теоретического материала и аналогов | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 10 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

В ходе выполнения дипломного проекта был изучен *IT*-рынок и выявлены аналоги разрабатываемого приложения. В результате чего в этой главе были описаны некоторые существующие чаты с поддержкой сквозного шифрования.

* 1. Постановка задачи

Целью дипломного проекта является разработка веб-приложения для обмена зашифрованными сообщениями.

Также в список возможностей приложения входят:

– регистрация и авторизация пользователей;

– возможность добавления пользователя в список контактов;

‒ удаление пользователя из списка контактов;

– обмен ключевой информации;

– пересылка сообщений без перезагрузки страницы;

– пересылка видео, документов, фотографий в зашифрованном виде;

– экспорт и импорт ключевой информации в файл;

– возможность установить пароль на файл при экспорте;

– поиск по сообщениям на стороне клиента;

– просмотр ключевой информации в виде набора *emoji*-смайлов;

Актуальность задачи

В последнее время вырос интерес к вопросам защиты информации. Это связывают с тем, что стали более широко использоваться вычислительные сети, что приводит к тому, что появляются большие возможности для несанкционированного доступа к передаваемой информации. Кроме того, чаты все больше вытесняют такие виды передачи информации как электронная почта, социальные сети, форумы. Прежде всего это связано с возрастанием спроса на передачу информации в «реальном времени”.

Большинство крупных компаний поставщиков сервисов передачи информации начали внедрять в свои продукты возможность шифрования на устройствах клиентов, начиная с 2010 года. Это связано с тем, что у многих людей появилось недоверия к надежности центров сертификации и сохранности своих данных на серверах сторонних компаний.

* 1. Описание сквозного шифрования

Сквозное шифрование гарантирует, что доступ к исходному тексту сообщения имеется только у отправителя и получателя. Это означает, что пользовательская информация становится недоступной даже серверам, передающим данные. Шифрование и дешифрование происходит на конечных устройствах пользователей. Кроме того, данные остаются зашифрованными, пока не будут доставлены к месту назначения. Поэтому часто сквозное шифрование также называют «нулевой доступ» или «шифрование на стороне клиента».

Более распространенная форма шифрования, известная как протокол защиты транспортного уровня, защищает сообщения по пути с вашего устройства на сервер мессенджера и с сервера на устройства получателя. Серверы поставщика услуг могут просматривать незашифрованные сообщения, хранить их и передавать в правоохранительные органы. Кроме того, конфиденциальная информация может попасть в руки злоумышленников при взломе серверов. Сквозное шифрование гарантирует, что никто не сможет прослушать, прочитать или просмотреть сообщение во время его передачи. Для того чтобы это сделать, преступники должны взломать устройство отправителя или получателя. В глобальных масштабах это происходит крайне редко.

* 1. Схема шифрования

Как говорилось выше, основной особенностью чата является сквозное шифрование. Сквозное шифрование предполагает, что процедура зашифровки и расшифровки сообщений происходит на клиентах и ключи для расшифровки хранятся на клиентах. Этот ключ используется для симметричного шифрования (т.е. одним ключом). Но для того, чтобы клиенты договорились какой ключ использовать, необходимо совершить процедуру “рукопожатия”, для этого я использую асимметричное шифрование.

Для реализации обмена общим секретом были выбраны алгоритмы *RSA*, *AES*.

*RSA* – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. *RSA*-ключи генерируются следующим образом:

1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое);
2. Вычисляется их произведение, называемое модулем;
3. Вычисляется значение функции Эйлера для числа n: ;
4. Выбирается целое число e, взаимно простое со значением функции Эйлера;
5. Вычисляется число d, мультипликативно обратное к числу e по модулю , т.е число удовлетворяющее условию: ;
6. Пара (e,n) является открытым ключом;
7. Пара (d,n) является закрытым ключом.

*Advanced Encryption Standard (AES)* - симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса *AES*. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником *DES*. Национальный институт стандартов и технологий США опубликовал спецификацию *AES* 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года *AES* был объявлен стандартом шифрования. По состоянию на 2009 год *AES* является одним из самых распространённых алгоритмов симметричного шифрования.

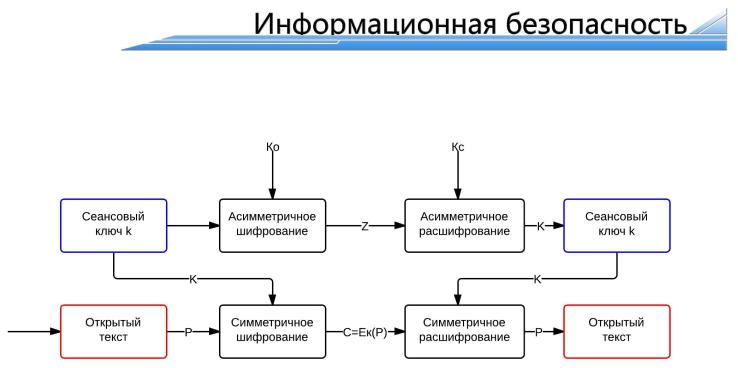


Рисунок 1.1 – Обмен ключом с помощью асимметричного шифрования.

* 1. Возможные атаки

Сквозное шифрование предусматривает, что контроль за перепиской осуществляется непосредственно пользователями. Одним из вариантов обхода сквозного шифрования для злоумышленника является захват под свой контроль канала связи между конечными точками, после этого он может попытаться выдать себя за получателя сообщения, чтобы, например, подменить открытый ключ. Чтобы не дать себя обнаружить, злоумышленник после дешифровки сообщения может зашифровать его ключом, который он разделяет с фактическим получателем, или его открытым ключом (в случае асимметричных систем) и снова отправить сообщение. Атаки такого типа принято называть атаками «человек посередине».

Для предотвращения *MITM-атак* большинство криптографических протоколов используют аутентификацию. Для этого могут использоваться, например, центры сертификации. Альтернативным методом является создание отпечатков открытого ключа на основе общедоступных открытых ключей пользователей или общих секретных ключей. Прежде чем начать разговор, стороны сравнивают свои отпечатки открытых ключей с использованием внешнего канала связи, который гарантирует целостность и аутентичность связи, при этом он не обязательно должен быть секретным. Если отпечатки ключей совпадают, значит атака «человек посередине» не была произведена.

Другим способом обхода сквозного шифрования является атака непосредственно на конечные точки доступа. Каждое устройство пользователя может быть взломано, с целью украсть криптографический ключ (для создания атаки «человек посередине») или просто прочитать дешифрованные сообщения пользователей. Для избежания такого рода попыток взлома, необходимо обеспечить соответствующую защиту пользовательских устройств с помощью программных или иных методов. Основными попытками повысить безопасность конечных точек были выделение ключевых операций генерации, хранения и криптографии на смарт-карту, например, в *Project Vault Google*. Тем не менее, так как ввод и вывод открытого текста видны в системе, то эти подходы не способны защитить от клавиатурных шпионов и вредоносного программного обеспечения, которое может отслеживать разговоры в режиме реального времени. Более надежный подход заключается в физической изоляции устройства.

Компании могут также (самостоятельно или по принуждению) внедрять в свое программное обеспечение бэкдоры, которые помогают нарушить согласование ключа или обойти шифрование.

* 1. Обзор аналогов

Существует множество чатов работающих как веб-приложение в браузере. Однако не все они имею возможность использовать шифрование на стороне клиента. Мною были найдены несколько аналогов, которые достаточно хорошо реализованы.

## Telegram

Для анализа прототипов было выбрано приложение *Telegram*. Проект создан Павлом Дуровым, основателем социальной сети *ВКонтакте*. Сервис построен на технологии шифрования переписки *MTProto*. Сам Telegram изначально был экспериментом принадлежащей Павлу компании *Digital* *Fortress* с целью протестировать *MTProto* на больших нагрузках.

*MTProto* ­– криптографический протокол, используемый в системе обмена сообщениями *Telegram* для шифрования переписки пользователей. Протокол был разработан Николаем Дуровым и другими программистами *Telegram*. В основе протокола лежит оригинальная комбинация симметричного алгоритма шифрования *AES* (в режиме *IGE*), протокол *Диффи*-*Хеллмана* для обмена 2048-битными *RSA*-ключами между двумя устройствами и ряд хеш-функций. Протокол допускает использование шифрования *end*-*to*-*end* с опциональной сверкой ключей. Также на основе данного протокола был создан *MTProxy*.

Сервис абсолютно бесплатен и в нём отсутствует реклама. Для тех же пользователей, кто заинтересован в максимальной конфиденциальности, *Telegram* предлагает Секретные чаты. Их можно запрограммировать на автоматическое самоуничтожение. Таким образом, можно быстро удалять все виды контента - сообщения, фотографии, видео и даже файлы. Секретные чаты применяют *E2EE*-*шифрование* для максимальной защиты информации.

Кроме веб-версии, приложение Telegram доступно на операционных системах *Linux*, *Windows*, *MacOS*, *Android*, *iOS*.

Интерфейс программы удобен и понятен, содержит минимальное количество анимации, что позволяет программе работать быстро на устройствах с минимальными техническими характеристиками.

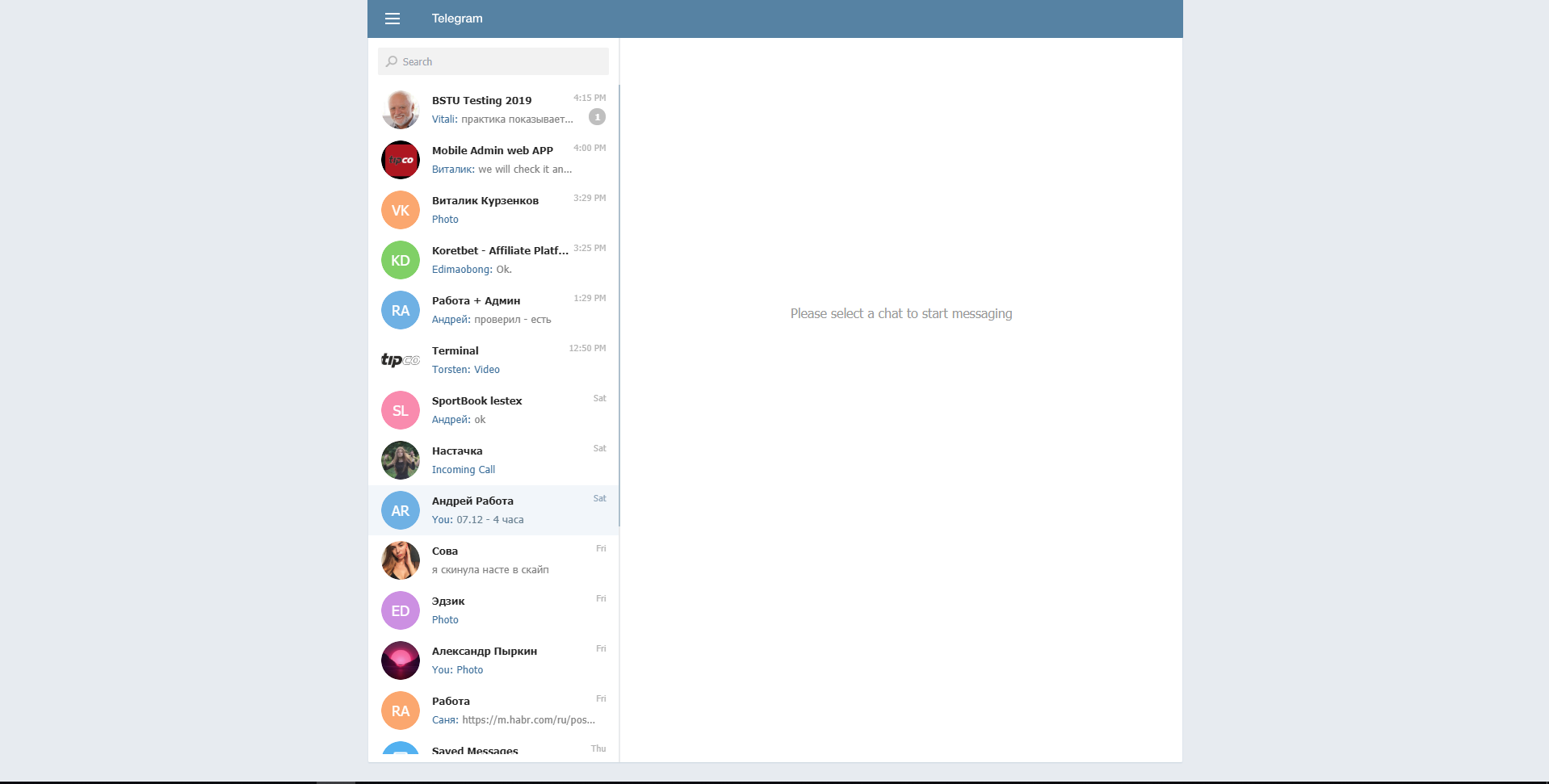


Рисунок 1.2 – Стартовая страница *Telegram*

В секретном чате пользователю доступна возможность просмотра секретного ключа шифрования. Для удобного сравнения ключей между пользователями, ключ представлен в виде набора квадратов разного цвета.

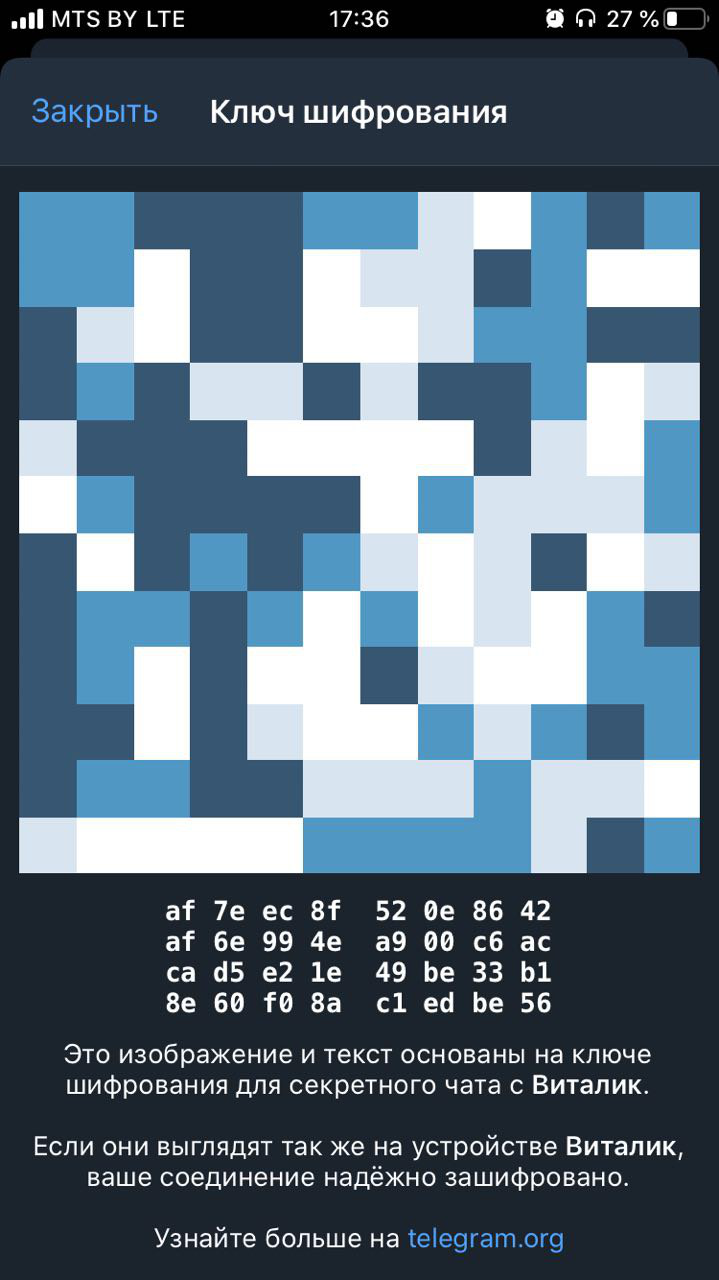


Рисунок 1.3 – Ключ шифрования в мобильной версии *Telegram*

*Telegram* является одним из самых популярных чатов в мире и «первопроходцем» в сфере сквозного шифрования для массовых пользователей.

## Whatsapp

*WhatsApp* – популярная бесплатная система мгновенного обмена текстовыми сообщениями для мобильных и иных платформ с поддержкой голосовой и видеосвязи. Позволяет пересылать текстовые сообщения, изображения, видео, аудио, электронные документы и даже программные установки через Интернет. Клиент работает на платформах *Android*, *iOS*, *Windows* *Phone*, *Windows* и в виде веб-приложения.

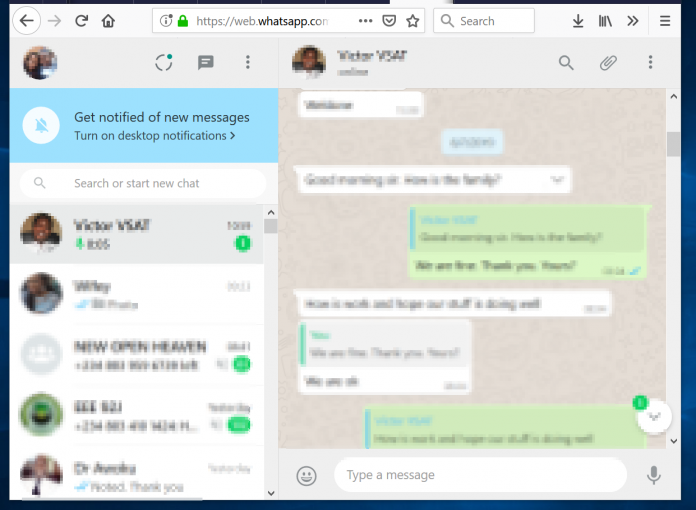


Рисунок 1.4 – Стартовая страницы *WhatsApp*

Приложение не с самого начала имело поддержку сквозного шифрования, только с апреля 2016 года *WhatsApp* включил сквозное шифрование для всех пользователей на базе разработок *Signal*. Шифрование распространяется на все типы сообщений: текст, фото, видео и голосовые сообщения. Шифрование также доступно в групповых чатах. По заявлениям компании расшифровать подобные сообщения может только получатель, содержимое недоступно даже серверам *WhatsApp*. В реализации используются алгоритмы *ECDH* на *Curve25519*, *AES*-*256*, *AES*-*GCM*, *HMAC*-*SHA256*, *HKDF*. Два пользователя могут сверить ключи шифрования путём сканирования QR кода или сравнения 60-значного числа, что позволит исключить атаки класса *man*-*in*-*the*-*middle*.

Мессенджер *WhatsApp* стал бесплатным с 18 января 2016 года. Ранее за подписку на использование сервиса взималась плата в размере около 1 доллара США каждый год, начиная со второго года использования, либо (для платформ *Apple*), однократный платёж при покупке приложения.

Основным отличием от секретных чатов *Telegram* является то, что сквозное шифрование включено по умолчанию для всех чатов. Это является несомненным плюсом, т.к. информация пользователя будет конфиденциально передаваться независимо от технического уровня подготовки.

Данный сайт обладает привлекательным дизайном, отличной скоростью работы и адаптивностью под все платформы.

На рисунке 1.5 можно видеть пример QR кода в версии *WhatsApp* для операционной системы *Android*.

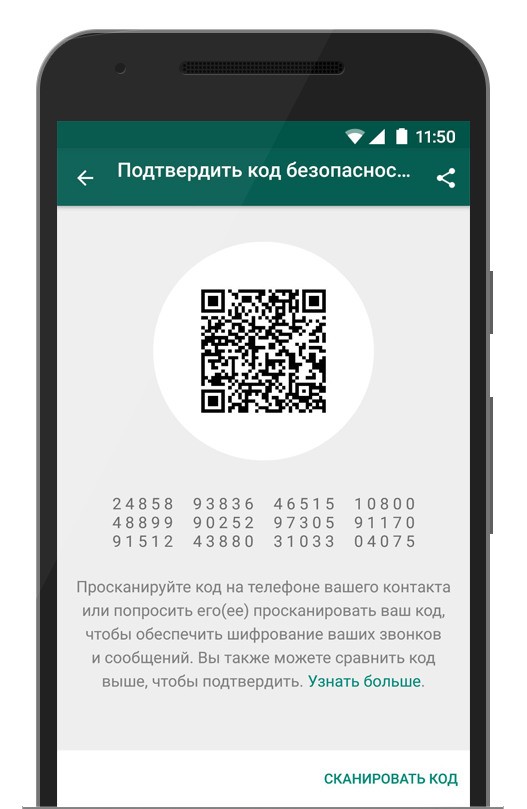


Рисунок 1.5 – QR код для проверки подлинности

* 1. Патентный поиск по теме дипломного проекта

В соответствии с темой дипломного проекта был проведен патентный поиск в области веб-приложений для учета хода выполнения дипломных проектов.

Патентный поиск ­– это процесс отбора соответствующих запросу документов или сведений по одному или нескольким признакам из массива патентных документов или данных, при этом осуществляется процесс поиска из множества документов и текстов только тех, которые соответствуют теме или предмету запроса.

Патентный поиск осуществляется посредством информационно-поисковой системы и выполняется вручную или с использованием соответствующих компьютерных программ, а также с привлечением соответствующих экспертов.

Предмет поиска определяют исходя из конкретных задач патентных исследований категории объекта (устройство, способ, вещество), а также из того, какие его элементы, параметры, свойства и другие характеристики предполагается исследовать.

При патентном поиске сравниваются выражения смыслового содержания информационного запроса и содержания документа.

Изобретение – это основной объект промышленной собственности.

Патент на изобретение – это документ, выдаваемый компетентным государственным органом и удостоверяющий: приоритет изобретения, авторство и исключительное право на изобретение. Действует в пределах территории того государства, ведомство которого его выдало.

Под патентоспособностью понимают юридическое свойство объекта промышленной собственности, определяющее его способность охраняться документом исключительного права (патентом) на территории конкретной страны в течение срока действия патента.

Решение является патентоспособным если:

* это способ или вещество;
* обладает новизной;
* имеет изобретательский уровень;
* является промышленно применяемым.

Патентный поиск – это процесс отбора соответствующих запросу документов или сведений по одному или нескольким признакам из массива патентных документов или данных, при этом осуществляется процесс поиска из множества документов и текстов только тех, которые соответствуют теме или предмету запроса.

В данном подразделе представлены результаты патентного исследования. В результате проведенного патентного поиска был выявлен ряд патентов, представленных в таблицах 1.1, 1.2, 1.3.

Таблица 1.1 – Описание патента №1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| СИСТЕМА ОБМЕ-НА СООБЩЕНИЯ-МИ МЕЖДУ ПОЛЬ-ЗОВАТЕЛЯМИ РАЗЛИЧНЫХ ВЕБ-САЙТОВ | WO2019132731 | 04.07.2019 | СКВОРЦОВ, Игорь Юрьевич SKVORTSOV, Igor Yurievich; RU  МАНШИРОВ, Рауль MANSHIROV, Raul; IL |

Группа изобретений относится к технологиям обмена сообщениями и/или звонками между пользователями различных веб-сайтов в сети Интернет с помощью технологии клиент-сервер. Техническим результатом является создание способа и самонастраивающейся интерактивной системы обмена сообщениями и/или звонками между пользователями веб-сайтов, которые обеспечивают возможность автоматизированного подбора собеседника для такого обмена, а также обеспечивают возможность автоматизированной блокировки обмена сообщениями и/или звонками с конкретным пользователем, в зависимости, по меньшей мере, от рейтинга и/или тематического рейтинга пользователя. Войдя в систему, пользователь может просматривать страницы различных пользователей, включая его друзей, оставлять комментарии на этих страницах, а также просматривать комментарии других пользователей на его странице, осуществлять личный и/или групповой обмен сообщениями и/или звонками между пользователями. В системе ведется анализ интересующих пользователя веб-страниц, аккаунтов пользователей. Для каждого пользователя вычисляются рейтинг и тематический рейтинг, которые отображают уровень доверия сообщениям, которыми обмениваются пользователи. В системе есть возможность автоматизированного предоставления для пользователя собеседника по обмену сообщениями и/или звонками, в зависимости от рейтинга, тематического рейтинга пользователя.

Таблица 1.2 – Описание патента №2

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| СПОСОБ И УСТРОЙСТВО ДЛЯ ОСУЩЕСТВЛЕНИЯ СВЯЗИ СО СКВОЗ-НЫМ ШИФРОВАНИЕМ | RU2495532C2 | 2013.10.10 | КАУХАНЕН Лари-Микко (FI) |

Изобретение относится к телекоммуникациям, а именно к способам осуществления связи со сквозным шифрованием. Техническим результатом является повышение безопасности передачи данных. Технический результат достигается тем, что способ осуществления связи со сквозным шифрованием, включает: хранение в пользовательском терминале комплекта ключей, пригодных для сквозного шифрования коммуникаций, при этом один или более ключей ассоциированы с индикатором действительности, определяющим криптографическую группу из двух или более пользовательских терминалов, в которой должен применяться соответствующий ключ, и период, в течение которого указанный ключ применим в указанной криптографической группе; подсоединение, на пользовательском терминале, к коммуникации криптографической группы и выбор подлежащего использованию ключа из хранящегося комплекта ключей на основе указанной криптографической группы и текущего времени в соответствии с ассоциированными индикаторами действительности.

Таблица 1.3 – Описание патента №3

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| СПОСОБ И СИСТЕМА ОБЕСПЕЧЕНИЯ УСЛУГ ГРУППОВОГО ИНТЕРАКТИВНОГО ОБЩЕНИЯ | RU2374771C2 | 2009.06.20 | ЦИНЬ Пэнчэн (CN),  ЛИНЬ Юяо (CN),  ФАНЬ Чжибинь (CN |

Изобретение относится к области сетей передачи данных, а более конкретно - к мгновенному обмену сообщениями. Технический результат заключается в снижении требований к полосе пропускания сети и к аппаратному обеспечению компьютеров. Сущность изобретения заключается в создании чат–группы, включающей первичный узел и один или несколько вторичных узлов; установлении *Р2Р*-соединения с каждым из вторичных узлов; приеме голосовых данных, переданных со вторичных узлов по *Р2Р*-соединениям; микшировании принятых голосовых данных с голосовыми данными первичного узла для формирования смешанных голосовых данных и их передаче первичным узлом на вторичные узлы по *Р2Р*-соединениям.

Настоящее изобретение относится к технологии мгновенного обмена сообщениями, в частности к способам и системам обеспечения услуг группового интерактивного общения в системе мгновенного обмена сообщениями.

Главной целью настоящего изобретения является разработка способа и системы обеспечения услуг группового чата, снижающих требования к полосе пропускания сети и к аппаратному обеспечению компьютеров для группового чата.

* 1. Выводы по разделу

В данном разделе был произведён обзор аналогов, рассмотрены их плюсы и минусы, произведен патентный поиск аналогов программного обеспечения по теме дипломного проекта. Каждый аналог был подробно описан, и было показано, для чего предназначено то или иное программное обеспечение.

Как видно из представленного краткого обзора, большинство ресурсов имеют много общего в своих функциональных возможностях. В первую очередь это касается базовых функций. Поэтому обычному пользователю иногда бывает трудно выбрать тот или иной ресурс.

Основным недостатком всех найденных аналогов является отсутствие сквозного шифрования в веб-версиях.

Это обусловлено несколькими факторами:

– необходимостью доверять веб-браузеру, на котором работает приложение;

– нагрузкой производительности браузера на вычисление ключей и шифрования;

– сложностью переноса ключей с устройства на другое устройство.

Как можно видеть приложения имеют некоторые недостатки, таким образом была выявлена целесообразность разработки и основной спектр функциональных возможностей, на которые в будущем необходимо сделать упор, для выделения приложения на фоне его конкурентов. Так же, на основании использования приложений-аналогов были выработаны основные критерии для построения дизайна приложения, позволяющего осуществлять наиболее простое и интуитивное пользование приложением.

1. Проектирование веб-приложения

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 02.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 2 Проектирование веб-приложения | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 9 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

* 1. **Обзор средств** разработки

## Платформа .NET Core

Для реализации данного веб-приложения была выбрана платформа *.NET Core*.

Платформа *.NET Core* – это платформа разработки общего назначения с открытым кодом, предназначенная для создания кроссплатформенных приложений. Вы можете создавать приложения *.NET Core* для *Windows*, *macOS* и *Linux* с поддержкой процессоров *x64*, *x86*, *ARM32* и *ARM64*, используя несколько языков программирования. Вам доступны платформы и *API-интерфейсы* для создания облачных приложений, приложений для Интернета вещей, использования клиентского интерфейса и машинного обучения.

*.NET Core* состоит из перечисленных ниже компонентов:

– cреда выполнения *.NET Core* предоставляет систему типов, функции загрузки сборок, сборщик мусора, собственные функции взаимодействия и другие базовые службы;

– cреда выполнения *ASP.NET Core* - платформа для создания современных облачных приложений, подключенных к Интернету: веб-приложений, приложений Интернета вещей, серверной части мобильных решений и многого другого;

– *NET Core CLI* и компиляторы языков (*Roslyn* и *F#*) реализуют возможности разработки *.NET Core*.

## Платформа ASP.NET Core

Платформа *ASP.NET Core* представляет технологию от компании *Microsoft*, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, *ASP.NET Core* является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход *ASP.NET Core* фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась *ASP.NET vNext*.

*ASP.NET Core* теперь полностью является *opensource*-фреймворком. Все исходные файлы фреймворка доступны на *GitHub*. *ASP.NET Core* может работать поверх кросс-платформенной среды *.NET Core*, которая может быть развернута на основных популярных операционных системах: *Windows*, *Mac OS*, *Linux*. И таким образом, с помощью *ASP.NET Core* мы можем создавать кросс-платформенные приложения. И хотя *Windows* в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС *Windows*, но и на *Linux* и *Mac OS*. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер *Kestrel*.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер *Nuget*. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку *System.Web.dll*.

*ASP.NET Core* включает в себя фреймворк *MVC*, который объединяет функциональность *MVC*, *Web API* и *Web Pages*. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель *ASP.NET Core MVC*. А *Web Forms* полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в *MVC* был добавлен ряд дополнительных функций. Одной из таких функций являются тэг-хелперы (*tag* *helper*), которые позволяют более органично соединять синтаксис *html* с кодом *С#*. *ASP.NET Core* характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как *Autofac*, *Ninject*. Хотя при желании их также можно продолжать использовать.

Для обработки запросов теперь используется новый конвейер *HTTP*, который основан на компонентах *Katana* и спецификации *OWIN*. А его модульность позволяет легко добавить свои собственные компоненты. Если суммировать, то можно выделить следующие ключевые отличия *ASP.NET Core* от предыдущих версий *ASP.NET*:

* новый легковесный и модульный конвейер *HTTP*-запросов;
* возможность развертывать приложение как на *IIS*, так и в рамках своего собственного процесса;
* использование платформы *.NET Core* и ее функциональности;
* распространение пакетов платформы через *NuGet*;
* интегрированная поддержка для создания и использования пакетов *NuGet*;
* единый стек веб-разработки, сочетающий *Web UI* и *Web API*;
* конфигурация для упрощенного использования в облаке;
* встроенная поддержка для внедрения зависимостей;
* расширяемость;
* кроссплатформенность: возможность разработки и развертывания приложений *ASP.NET* на *Windows*, *Mac* и *Linux*;
* развитие как open source, открытость к изменениям.

## Среда разработки

В качестве среды разработки было решено использовать *Microsoft Visual Studio 2019,* т.к. она включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. *Visual Studio* позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, *Subversion* и *Visual SourceSafe*), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент *Team Explorer* для работы с *Team Foundation Server*).

## **Обоснование** выбора языка программирования

*C#* – язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998–2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании *Microsoft* как основной язык разработки приложений для платформы *Microsoft* .*NET*. Компилятор с *C#* входит в стандартную установку самой .*NET*, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде *Visual Studio*.

*C#* относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к *C++* и *Java*. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников – языков *C++*, *Delphi*, *Modula* и *Smalltalk* – *С#*, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, *C#* не поддерживает множественное наследование классов (в отличие от *C++*) или вывода типов (в отличие от *Haskell*). С помощью языка *C#* можно создавать обычные приложения *Windows*, *XML*-веб-службы, распределенные компоненты, приложения «клиент-сервер», приложения баз данных и т. д.

Исходя из перечисленных преимуществ, для разработки веб-приложения был выбран язык *C#* [1]*.*

## **Программное** обеспечение сервера

Обработка *НТТР*-запросов – это основная задача в большинстве сетевых программ. Для выполнения данной задачи существует специальный вид программного обеспечения – веб-серверы.

Сервер *IIS Express* имеет многие функции веб-сервера *IIS 7* и *IIS 8*, для использования которых не требуется устанавливать на компьютер службы *IIS* и управлять ими. Поскольку сервер *IIS Express* похож на сервер *IIS 7* и *IIS 8*, то по сравнению с использованием других веб-серверов при его применении переводить веб-сайты на сервер *IIS* проще.

Сервер *IIS Express* реализует следующие возможности:

* он поддерживает ту же модель расширения и параметры файла *Web.config*, что и службы *IIS 7* и *IIS 8*;
* не требует изменения кода веб-приложения;
* его можно установить параллельно на одном компьютере с полной версией служб IIS и другими веб-серверами.

При разработке веб-проектов в *Visual Studio* требуется веб-сервер для их тестирования и запуска. *Visual Studio* позволяет выполнять тестирование на различных веб-серверах, в том числе на *IIS Express*, сервере служб *IIS*, внешних и пользовательских веб-серверах. Для проекта веб-приложения на основе файлов можно использовать любой из этих серверов. Для проектов веб-сайта на основе файлов можно использовать *IIS Express*.

*IIS Express* является веб-сервером по умолчанию для проектов веб-приложений в *Visual Studio 2012* и *Visual Studio 2013*. Его рекомендуется использовать как для проектов веб-приложений, так и для проектов веб-сайтов. Поэтому для разработки веб-приложения был сервером для тестирования был выбран *IIS Express*.

## База данных MS SQL Server

*Microsoft SQL Server* — система управления реляционными базами данных (РСУБД), разработанная корпорацией *Microsoft*. Основной используемый язык запросов — *Transact-SQL*, создан совместно *Microsoft* и *Sybase. Transact-SQL* является реализацией стандарта *ANSI/ISO* по структурированному языку запросов (*SQL*) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

*MS SQL Server* – это платформа для решения критически важных задач в масштабе предприятия, обладающая высокой доступностью, повышенной производительностью и безопасностью. Решение представляет собой хорошо масштабируемый, полностью реляционный, быстродействующий сервер, способный обрабатывать большие объемы данных для клиент-серверных приложений. Рекордная производительность *MS SQL Server* обеспечивается новыми технологиями работы с памятью, что помогает предприятиям ускорить свой бизнес и реализовать новые сценарии работы. Кроме того, *SQL Server* позволяет использовать новые гибридные облачные решения и пользоваться новыми преимуществами облачных вычислений. Расширенные функции безопасности, в сочетании со встроенными, удобными для использования инструментами и управляемым доступом к данным, позволяют организации выполнить требования строгих политик соответствия нормам.

## Entity Framework

Центральной концепцией *Entity Framework* является понятие сущности или *entity*. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа *int*, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Существует три подхода к работе с данными в *Entity Framework*: *Database First*, *Model First*, и *Code First* представленных на рисунке 2.3.

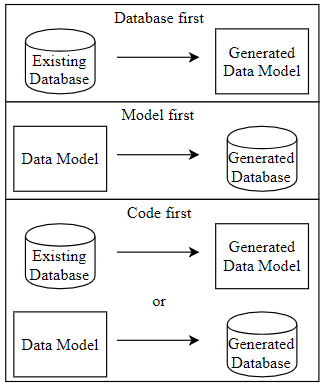


Рисунок 2.1 – Подходы к разработке с *Entity Framework*

*Database First*. В случае уже имеющейся базы данных *Entity Framework* может автоматически создать модель данных, состоящую из классов и свойств, соответствующих объектам базы данных (таким, как таблицы и столбцы). Информация о структуре базы (*store schema*), модель данных (*conceptual model*) и маппинг их друг на друга содержится в *XML* в файле *.edmx*. *Visual Studio* предоставляет графический дизайнер *Entity Framework*, с помощью которого можно просматривать и редактировать *.edmx*.

*Model First*. Когда базы нет, вы можете начать с создания модели данных, используя дизайнер *Entity Framework Visual Studio*. После окончания работ над моделью дизайнер сгенерирует *DDL* (*data definition language*)-код для создания базы. В этом подходе для хранения информации о модели и маппингах также используется *.edmx*.

*Code First*. Вне зависимости от наличия базы вы можете вручную написать код классов и свойств, соответствующих сущностям в базе и использовать этот код с *Entity Framework* без использования файла *.edmx*. Именно поэтому можно порой увидеть, как этот подход называют *code only*, хотя официальное наименование *Code First*. Если базы ещё нет, *Entity Framework* может создать, удалить или пересоздать её в случае изменений в модели.

*API* доступа к данным, разработанное для *Code First*, основано на классе *DbContext*. API может быть использован также и в процессе разработки с подходами *Database First* и *Model First*.

## Angular

*Angular* представляет фреймворк от компании *Google* для создания клиентских приложений. Прежде всего он нацелен на разработку *SPA*-решений (*Single Page Application*), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка *AngularJS*. В то же время *Angular* это не новая версия *AngularJS*, а принципиально новый фреймворк. *Angular* предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей *Angular* является то, что он использует в качестве языка программирования *TypeScript*. Поэтому перед началом работы рекомендуется ознакомиться с основами данного языка. Но мы не ограничены языком *TypeScript*. При желании можно писать приложения на *Angular* с помощью таких языков как *Dart* или *JavaScript*. Однако *TypeScript* все таки является основным языком для *Angular* [12].

## Язык TypeScript

*TypeScript* представляет из себя язык программирования на основе *JavaScript*. Его развитие началось в конце 2012 года. Хотя он зародился в компании *Microsoft*, и его фактическим создателем является программист Андерс Хейлсберг, так же известный как создатель таких языков как *Delphi*, *C#*, но данный проект сразу стал развиваться как *OpenSource*. И уже с самого начала новый язык стал быстро распространяться в силу своей гибкости и производительности. Немало проектов, которые были написаны на *JavaScript*, стали переноситься на *TypeScript*. Популярность и актуальность идей нового языка привела к тому, что ряд из этих идей в последующем станут частью нового стандарта *JavaScript*. А новая версия одного из популярнейших фреймворков для *Web* - *Angular 2/4/5/6* полностью написана на *TypeScript* совместно компаниями *Microsoft* и *Google*.

## SignalR

*SignalR Core* представляет библиотеку от компании *Microsoft*, которая предназначена для создания приложений, работающих в режиме реального времени. В частности, ее можно использовать вместе с *ASP.NET Core*. *SignalR* использует двунаправленную связь для обмена сообщениями между клиентом и сервером, благодаря чему сервер может отправлять в режиме реального времени всем клиентам некоторые данные. *SignalR* используется в приложениях, которые получают данные в реальном режиме времени, например, чаты, социальные сети, игровые приложения, карты, приложения для аукционов, голосований и карт, панели управления, приложения для мониторинга данных и так далее.

Исходя из возможностей клиента и сервера инфраструктура *SignalR* выбирает наилучший механизм для взаимодействия. В частности, наиболее оптимальным является *WebSockets*, соответственно если и клиент, и сервер позволяют использовать этот механизм, то взаимодействие идет через *WebSockets*. Однако если *WebSockets* не поддерживается, то применяется *Server-Side Event*s. И если *SSE* не поддерживается, то применяется *Long Polling* [4].

## JavaScript

*JavaScript* – это полноценный динамический язык программирования, который применяется к *HTML* документу, и может обеспечить динамическую интерактивность на веб-сайтах. Его разработал *Brendan Eich*, сооснователь проекта *Mozilla*, *Mozilla Foundation* и *Mozilla Corporation*.

*JavaScript* довольно универсален. Работу с данным языком можно начать с малого, с простых функций, таких как карусели, галереи изображений, изменяющиеся макеты и отклик на нажатие кнопок. Обладая большим опытом уже можно создавать игры, анимированную *2D* и *3D* графику, полномасштабные приложения с базами данных и многое другое.

*JavaScript* сам по себе довольно компактный, но очень гибкий. Разработчиками написано большое количество инструментов поверх основного языка *JavaScript*, которые разблокируют огромное количество дополнительных функций с очень небольшим усилием. К ним относятся:

– программные интерфейсы приложения (*API*), встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание *HTML* и установку *CSS* стилей, захват и манипуляция видеопотоком, работа с веб-камерой пользователя или генерация *3D* графики и аудио сэмплов;

– сторонние API позволяют разработчикам внедрять функциональность в свои сайты от других разработчиков, таких как *Twitter* или *Facebook*;

– также есть возможность применить к *HTML* сторонние фреймворки и библиотеки, что позволит ускорить создание сайтов и приложений.

Архитектура программного продукта

Графическое представление архитектуры проектируемого программного продукта изображено на рисунке 2.1.

Backend

Front

Angular

DB

WebSocket

Http

EF Core

Рисунок 2.1 – Архитектура системы

Архитектуру системы можно назвать *SPA ­­­(Single Page Application)*. В качестве сервера выступает *ASP.NET Cor*e, который на первый запрос клиента возвращает *Angular* приложение. После этого *Angular* клиент генерирует запросы, и с помощью технологии *Ajax*, отправляет их на сервер. Кроме это *Angular* подключается к серверу по технологии *WebSocket*. Сервер *ASP.NET* принимает запросы и с помощью *Entity Framework* взаимодействует с базой данных. Отличительной особенностью такой архитектуры является то, что клиентской части приложение не требуется полной перезагрузки страницы.

Для авторизации и аутентификации пользователя было выбрано решение *ASP.NET Core Identity*. По умолчанию данное решение создает ряд таблиц в выбранной базе данных и на основе этих таблиц создает модели в *Entity Framework*.

На рисунке 2.2 можно видеть, какие таблицы автоматически создает *ASP.NET Identity*.

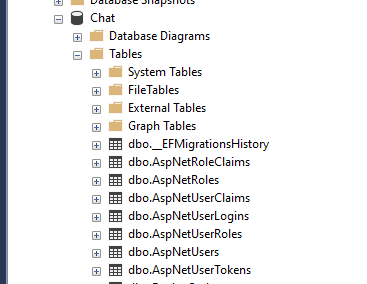


Рисунок 2.2 – Таблицы ASP.NET Identity

Кроме этого для аутентификации запросов на основе данных *ASP.NET Identity*, сервер отсылает клиентам *JWT-токен*. А клиент в свою очередь отсылает серверу этот токен с каждым запрос. Данная схема называется *JWT-аутентификация*.

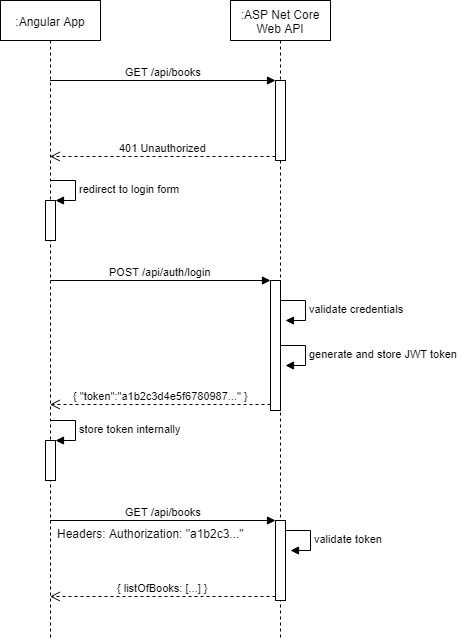


Рисунок 2.3 – Пример JWT-аутентификации

* 1. **Выводы** по разделу

В результате обзора технических средств и анализа поставленной задачи для реализации дипломного проекта была выбрана платформа *.NET Core*, так как данная платформа предоставляет весь нужный функциональный набор для создания веб-ресурса. Исходя из выбранной платформы, языком, для написания веб-ресурса был выбран *C#*, так как данный язык программирования позволяет создавать различные безопасные и надежные приложения, работающие на платформе *.NET Core*. В качестве СУБД был выбран *MS SQL Server*, позволяющий создать базу данных которая в дальнейшем использовалась для создания модели *ADO.NET EDM* в среде разработки *MS Visual Studio.* В основу работы с базой данных была выбрана технология конструктора EF. В качестве архитектуры приложения была выбрана *SPA (Single Page Application)*, сервер на *ASP.NET Core* на первый запрос передает клиенту Angular приложение, далее сервер работает как *WebApi* для Angular клиента.

1. Разработка программного обеспечения веб-приложения

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 03.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 3 Разработка программного обеспечения веб-приложения | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 17 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

* 1. Проектирование программного средства

Главной особенностью данного проекта является сквозное шифрование. Для реализации данной функции необходимо решить ряд задача:

* решить проблему ресурсозатратности алгоритмов шифрования в браузере;
* выбрать наиболее подходящее хранилище ключей;
* спроектировать формат хранения ключей;

Для решения первой задачи я воспользовался *WebCryptoApi*, которая реализована большинством современных браузеров. Интерфейс *Crypto* представляет базовые криптографические функции, доступные в общеупотребительном контексте. Он позволяет получить доступ к криптографически сильному генератору случайных чисел и к криптографическим примитивам. Исполнением и генерации ключей занимается не движок браузера, а непосредственно процесс браузера. Именно это *Api* позволит решить проблему производительности. В сравнении с криптографическими библиотеками, который выполняются в браузере в виде *Javascript*, данное решение обладает колоссальным преимуществом в производительности.

Объект с этим интерфейсом доступен в *Web*-контексте через свойство *window.crypto*.

В качестве алгоритмов шифрования были выбраны *RSA-OAEP* и *AES-CTR*. Схема обмена общим секретом была описана в теоретическом описании. Для реализации *WebCryptoApi*, я создал отдельный сервис, в котором реализовал все методы шифрования и дешифрования.

public async generateRSAKeys() {

return await window.crypto.subtle.generateKey(

{

name: "RSA-OAEP",

modulusLength: 2048,

publicExponent: new Uint8Array([0x01, 0x00, 0x01]),

hash: { name: "SHA-256" },

},

true,

["encrypt", "decrypt"]

)

}

Листинг 3.1 – Пример реализации генерации ключа RSA

Все функции реализованы асинхронно, т.е выполнение их не будет останавливать основной поток браузера.

В качестве хранилища ключей был выбрано *Api* браузера *localStorage*. Данное *Api* позволяет хранить данные в формате «ключ-значение». Для формирования ключа были использованы *GUID* пользователей записанные через знак «||». Таким образом появляется возможность хранить ключи для нескольких чатов. В значения для каждого ключа записываются данный в формате *JWK(JsonWebKey)*. Данный формат является стандартным для библиотеки типов *WebCryptoApi*.

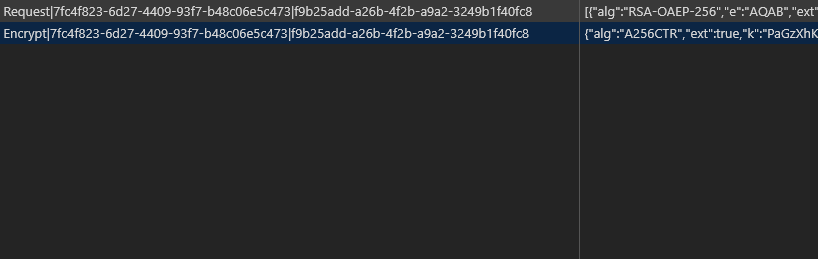


Рисунок 3.1 – Пример формата ключей в localStorage

Как можно видеть на рисунке 3.1, кроме GUID пользователей записывается ключевое слово «Request» либо «Encrypt», это позволяет различать какой ключ используется для обмена, а какой для шифрования. На рисунке 3.2 можно видеть все свойства формата *JWK*.

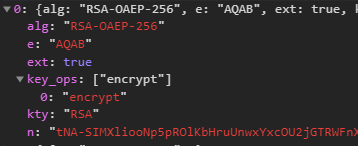


Рисунок 3.2 – Пример формата «JWK»

Основные проблемы проектирования данного проекта были решены. Универсальность решений позволяет запускать приложения на всех браузерах, поддерживающих *WebCryptoApi*.

* 1. Файловая структура

## WebApi

Перед началом разработки была продумана файловая структура проекта в соответствии с паттерном проектирования *MVC*, позволившая быстрее ориентироваться в коде, вносить изменения, дописывать новый функционал. Схема файловой структуры представлена на рисунке 3.3.

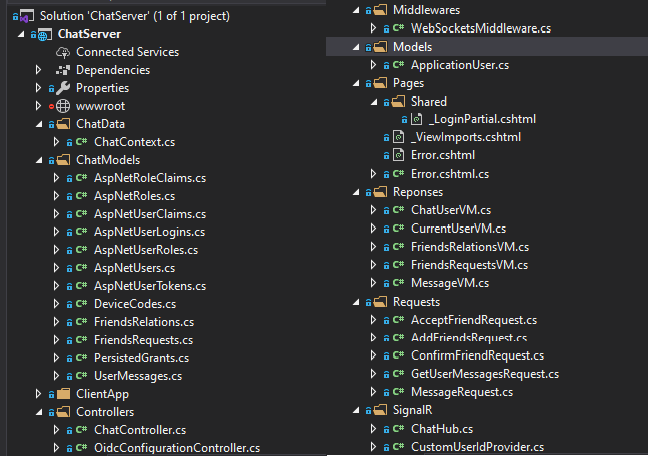


Рисунок 3.3 – Файловая структура *WebApi*

Основное решение проекта содержит большое количество папок, каждая из которых имеет свое назначение. Можно выделить несколько файлов отличающихся от стандартной структуры таких проектов. Один из таковых это файл *WebSocketMiddleware.cs*. Он необходим для того, чтобы перехватывать каждый запрос по протоколу *wss* и добавлять в заголовок запись токена авторизации. Другим файлом, отличающимся от стандартных, является *ChatHub.cs*. В файле описан веб-сокет библиотеки *SignalR*, называющийся *ChatHub*. Задача этого хаба – принимать сообщения по протоколу «wss», определять пользователя, который прислала это сообщение, и перенаправлять сообщение другом пользователю, указанному в параметре. В отличие от стандартных *WebApi*, данное приложение имеет папку *ClientApp*, в которой содержится клиентское приложение *Angular*. Файлы данного приложение «собирается» в папку *wwwroot*, и далее сервер возвращает эти файлы на запрос без параметров.

На листинге 3.2 можно увидеть реализацию класса *WebSocketMiddleware.cs*.

namespace ChatServer.Middlewares

{

public class WebSocketsMiddleware

{

private readonly RequestDelegate \_next;

public WebSocketsMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext httpContext)

{

var request = httpContext.Request;

if (request.Path.StartsWithSegments("/chatHub", StringComparison.OrdinalIgnoreCase) &&

request.Query.TryGetValue("access\_token", out var accessToken))

{

request.Headers.Add("Authorization", $"Bearer {accessToken}");

}

await \_next(httpContext);

}

}

}

Листинг 3.2 – реализация класса *WebSocketMiddleware.cs*

Также на рисунке 3.3 можно увидеть реализацию класса *ChatHub.cs*.

namespace ChatServer.SignalR

{

[Authorize]

public class ChatHub : Hub

{

public async Task Send(string message, string to, short type)

{

await Clients.User(to).SendAsync("Receive", message, type);

}

}

}

Листинг 3.3 – реализация класса *ChatHub.cs*

## Angular app

В данном подразделе я опишу структуру клиентского приложения и покажу некоторые особенности, обусловленные задачей приложения. Как говорилось выше, клиентское приложение находится в папке *ClientApp*. Далее структура разделятся на две подпапки. Первая папка – *node\_modules*, в ней содержатся все дополнения, которые были установлены по умолчанию *Angular* и которые ставились в процессе разработки. Следующая папка называется *src,* в ней хранятся все пользовательские компоненты, сервисы, модели, модули и т.д.

Главным конфигурационным файлом проекта является «angular.json». Он содержит в себе информацию о сборке, версии проекта, отладочной конфигурации, файловой структуре проекта.

На рисунке 3.4 можно видеть структуру клиентского приложения *Angular*.

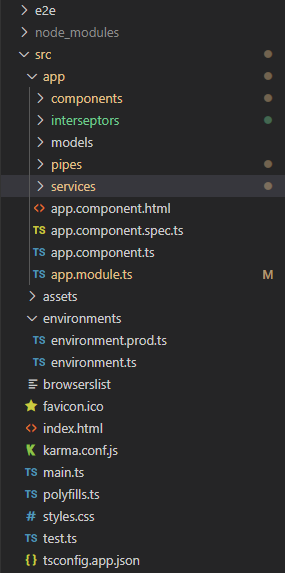


Рисунок 3.4 – Файловая структура клиентского приложения *Angular*

* 1. Структура базы данных

В результате проектирования проекта была создана база данных, которая удовлетворяет поставленным задачам. Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных. Для реализации функций, которые были определены на стадии проектирования, были созданы следующие таблицы:

* *AspNetUsers* (информация о пользователях);
* *UserMessages* (сообщения пользователей);
* *FriendsRelations* (информация о друзьях пользователей);
* *FriendRequest* (заявки в друзья пользователей);
* *AspNetUserLogins* (история авторизации пользователей);
* *AspNetRoles* (роли пользователей);
* *AspNetUserRole* (отношение ролей к пользователям);
* *AspNetUserClaims* (возможные политики авторизации и отношения с пользователями);
* *AspNetRoleClaims* (отношения ролей и политик);
* *AspNetUserTokens* (токены пользователей).

Логическая схема базы данных со всеми связями показана на рисунке 3.5 и в приложении А.

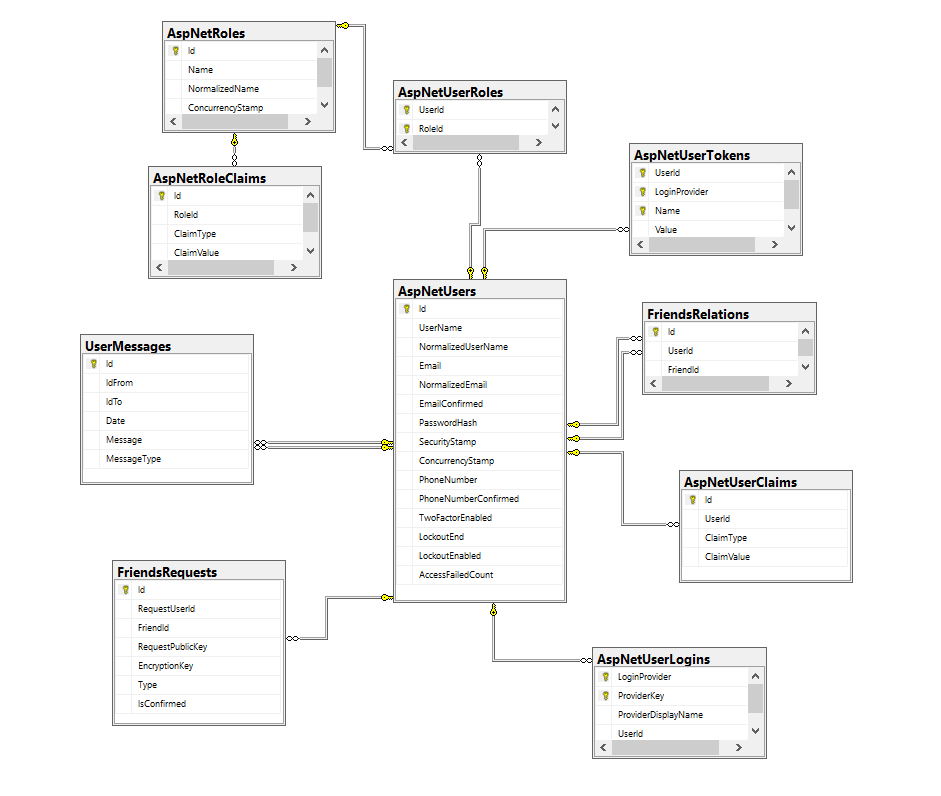


Рисунок 3.5 – Логическая схема базы данных

Для реализации функционала веб-приложения было создано десять таблиц. Далее приведена структура основных таблиц и их описание.

Таблица *AspNetUsers* содержит *GUID* пользователя, хэш от пароля, *email* пользователя, номер телефона, поле блокировки, поле подтверждения *email*. Данная таблица была сгенерирована автоматически, некоторые ее поля не используются, но могу быть использованы в дальнейшем при расширении приложения.

Описание ее полей приведено в таблице 3.1.

Таблица 3.1 – Структура таблицы *AspNetUsers*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | nvarchar(450) | Идентификатор пользователя |
| UserName | nvarchar(256) | Имя пользователя |
| NormalizedUserName | nvarchar(256) | Имя в верхнем регистре |
| Email | nvarchar(256) | Почта пользователя |
| NormalizedEmail | nvarchar(256) | Почта в верхнем регистре |
| EmailConfirmed | bit | Подтвержден ли email |
| PasswordHash | nvarchar(MAX) | Хэш пароля |
| SecurityStamp | nvarchar(MAX) | Печать безопасности |
| ConcurrencyStamp | nvarchar(MAX) | Печать параллелизма |
| PhoneNumber | nvarchar(MAX) | Телефонный номер |
| PhoneNumberConfirmed | bit | Подтвержден ли номер |
| TwoFactorEnabled | bit | Двухфакторная аутентификация |
| LockoutEnd | datetimeoffset(7) | Окончание блокировки |
| LockoutEnabled | bit | Включена ли блокировка |
| AccessFailedCount | int | Кол-во неудачных попыток входа |

Таблица *UserMessages* содержит идентификатор сообщения, имя отправителя, имя получателя, а также дату отправления, идентификатор типа, данные сообщения. Описание ее полей приведено в таблице 3.2.

Таблица 3.2 – Структура таблицы *UserMessages*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | int | Идентификатор сообщения |
| IdFrom | nvarchar(450) | Идентификатор отправителя |
| IdTo | nvarchar(450) | Идентификатор получателя |
| Date | datetime | Дата отправления |
| Message | nvarchar(MAX) | Данные сообщения |
| MessageType | smallint | Тип сообщения |

Таблица *FriendsRequests* содержит идентификатор заявки, идентификатор отправителя, идентификатор получателя, публичный ключ отправителя, ключ шифрования. Структура представлена в таблице 3.3.

Таблица 3.3 – Структура таблицы *FriendsRequests*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | int | Идентификатор заявки |
| RequestUserId | nvarchar(450) | Идентификатор заявителя в друзья |
| FriendId | Nvarchar(50) | Идентификатор получателя заявки |
| RequestPublicKey | nvarchar(MAX) | Публичны ключ заявителя |
| EncryptionKey | nvarchar(MAX) | Ключ шифрования |
| Type | smallint | Тип заявки |
| IsConfirmed | bit | Подтверждена ли заявка |

Таблица *FriendsRelations* содержит идентификатор, идентификаторы заявителя в друзья, а также идентификатор получателя заявки. Структура представлена в таблице 3.4.

Таблица 3.4 – Структура таблицы *FriendsRelations*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | int | Идентификатор отношений |
| UserId | nvarchar(450) | Идентификатор отправителя заявки |
| FriendId | nvarchar(450) | Идентификатор получателя заявки |

Выше были представлены таблицы, которые были созданы для функционирования всей системы веб-ресурса. Логическая схема базы данных представлена в приложении А.

* 1. Разработка контроллера ChatController

Основной функционал сервера содержится в контроллере *ChatConrtoller*. Это связано со спецификой разрабатываемой программы. Классический паттерн *MVC* предполагает наличие отдельного контроллера на каждое представление, но в случае, когда сервер возвращает только одну страницу *Angular*, а далее работает как простое *WebApi,* в этом нет необходимости.

Пример программного кода контроллера представлен на листинге 3.4.

public class ChatController : ControllerBase

{

IHubContext<ChatHub> HubContext { get; set; }

public ChatController(IHubContext<ChatHub> hubcontext)

{

this.HubContext = hubcontext;

}

[HttpGet("GetFriends")]

public IEnumerable<FriendsRelationsVM> GetFriends()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var req = db.FriendsRelations.Where((fr) => fr.UserId == userId).ToList().Select((fr) => new FriendsRelationsVM { Id = fr.Id, FriendId = fr.FriendId, UserId = fr.UserId });

return req;

}

}

}

Листинг 3.4 – Программная реализация контроллера *ChatController*

Основной функционал контроллера:

* получение списка заявок;
* получение списка неподтвержденных заявок;
* добавление новой заявки;
* подтверждение заявки;
* получение списка друзей;
* синхронизация локальных ключей и ключей базы данных;
* получение зашифрованных сообщений;
* добавление нового зашифрованного сообщения.

Как можно видеть контроллер взаимодействует с базой данных с помощью запросов *LINQ to Entities*. Все методы контроллера доступны только для авторизированных пользователей. Сервер проверяет заголовки запросов, эта опция доступна для разработчика с помощью атрибута *Authorize*.

Разработка приложения Angular

## Сервисы

Как говорилось выше, основная логика приложения будет реализована на стороне клиента. Для взаимодействия клиентского приложение с сервером использовалась библиотека *RxJs*, с помощью этой библиотеки можно отправлять запросы на сервер в асинхронном режиме. Для реализации этой библиотеки был выделен отдельный сервис *api.service.ts*.

export class ApiService {

constructor(private http: HttpClient, private config: ConfigService) {

}

getFriendsRequests(): Observable<FriendsRequest[]> {

return this.http.get<FriendsRequest[]>(`${this.config.app.api.url}Chat/GetFriendsRequests`);

}

getNotConfirmedRequests(): Observable<FriendsRequest[]> {

return this.http.get<FriendsRequest[]>(`${this.config.app.api.url}Chat/GetNotConfirmedRequests`);

}

getFriends(): Observable<FriendsRelations> {

return this.http.get<FriendsRelations>(`${this.config.app.api.url}Chat/GetFriends`);

}

getAllUsers(): Observable<ChatUserResponse[]> {

return this.http.get<ChatUserResponse[]> (`${this.config.app.api.url}Chat/GetAllUsers`);

}

getCurrentUser(): Observable<CurrentUserResponse> {

return this.http.get<CurrentUserResponse> (`${this.config.app.api.url}Chat/GetCurrentUser`);

}

}

Листинг 3.5 – Программная реализация сервиса *api.service.ts*

Также вся общая логика для компонент была выделена в *common.service*. Основной задачей этого сервиса является взаимодействие с локальным хранилищем.

Основные функции данного сервиса:

* синхронизация базы данных с локальным хранилищем;
* подтверждение и запись в локальное хранилище публичного ключа;
* подтверждение и запись в локальное хранилище ключа шифрования.

Функция синхронизации локального хранилища и базы данных представлена на листинге 3.6.

export class CommonService {

public onSelectedFriend = new Subject<ChatUserResponse>();

public onSelectedFriend$ = this.onSelectedFriend.asObservable();

public currentUser: CurrentUserResponse;

public enteredText: string;

constructor(private api: ApiService, private crypto: CryptoService) {

}

public synchronizationLocal(requests: FriendsRequest[]) {

const localItems = Object.keys(localStorage);

localItems.forEach((item) => {

const localItem = localStorage.getItem(item);

const splitIds = localItem.split('|');

if (splitIds && splitIds.length == 3 && !requests.find((req) => (req.requestUserId === splitIds[1] && req.friendId === splitIds[2])

|| (req.requestUserId === splitIds[2] && req.friendId === splitIds[1]))) {

localStorage.removeItem(item);

}

});

}

}

Листинг 3.6 – Программная реализация сервиса *common.service.ts*

Реализация алгоритмов шифрования описана в сервисе *crypto.service*. Кроме шифрование здесь также описаны переводы строк к типу *ArrayBuffer*, представление строк к формату *base64*, представление строк в виде *emoji*-символов. Также стоит выделить функции присвоения и разрешения пароля к строке. Данные функции используются при экспорте и импорте ключевой информации в файл. Эти методы, так же, как и методы шифрование реализуют библиотеку *WebCryptoApi*, разница заключается только в том, что функции для экспорта и импорта используют в качестве ключа пароль. Алгоритмом шифрования при экспорте был выбран *AES-GCM*. Большинство функции реализованы в асинхронном виде, т.е не останавливают основной поток. На листинге 3.7 можно видеть реализацию функции получения emoji-символов из строки.

public async getEmojiFromString(inputString: string, hashLength: number = 1) {

const hash = await window.crypto.subtle.digest('SHA-256', this.stringToArray(inputString));

const hexHash = this.buf2hex(hash);

const decimalHash = parseInt(hexHash, 16);

let emojiIndex = decimalHash % Math.pow(emojis.length, hashLength);

let emojiString = '';

for (let ii = 0; ii < hashLength; ii++) {

emojiString = `${emojis[emojiIndex % emojis.length]}${emojiString}`;

emojiIndex = Math.floor(emojiIndex / emojis.length);

}

return emojiString;

}

Листинг 3.7 – Программная реализация метода *getEmojiFromString*

Также в качестве важного сервиса можно рассмотреть интерцептор *AuthorizeInterceptor.* *Angular HTTP Interceptor* позволяет перехватывать HTTP-запросы перед их отправкой и вносить в них необходимые изменения. То же самое справедливо и для ответов сервера. В проекте интерцептор необходим для того, чтобы дописывать к запросам на сервер заголовок *Authorization* с токеном пользователя. Пример метода *processRequestWithToken* можно видеть на листинге 3.8.

private processRequestWithToken(token: string, req: HttpRequest<any>, next: HttpHandler) {

if (!!token && this.isSameOriginUrl(req)) {

req = req.clone({

setHeaders: {

Authorization: `Bearer ${token}`

}

});

}

return next.handle(req);

}

Листинг 3.7 – Программная реализация метода *processRequestWithToken*

## Компоненты

Основной компонентой приложения является *ChatComponent*. В ней содержится логика отправки, демонстрации сообщений. Компонента изменяется в зависимости от выбранного пользователя. Также компонента отвечает за постепенную подгрузку сообщений, по мере скроллинга. Данный функционал называется *Lazy Loading*. На листинге 3.8 модно видеть реализацию этой функции.

onScroll(event: any) {

if (this.messages && this.messages.length && !this.isLoadingMessages && event.target.scrollTop == 0) {

this.getMessagesAfter(this.messages[0]);

}

}

ngAfterViewInit() {

const chatContainer = document.getElementById("chat-container");

if (chatContainer) {

const scrollableContainer = chatContainer.querySelector('.scrollable');

scrollableContainer.addEventListener('scroll', (event) => { this.onScroll(event) });

}

}

Листинг 3.8 – Программная реализация *Lazy Loading*

Также данная компонента обладает функционалом *drag and drop*. Это позволяет пользователям перетаскивать файлы и отправлять их. На рисунке 3.6 представлен общий вид компоненты *ChatComponent*.

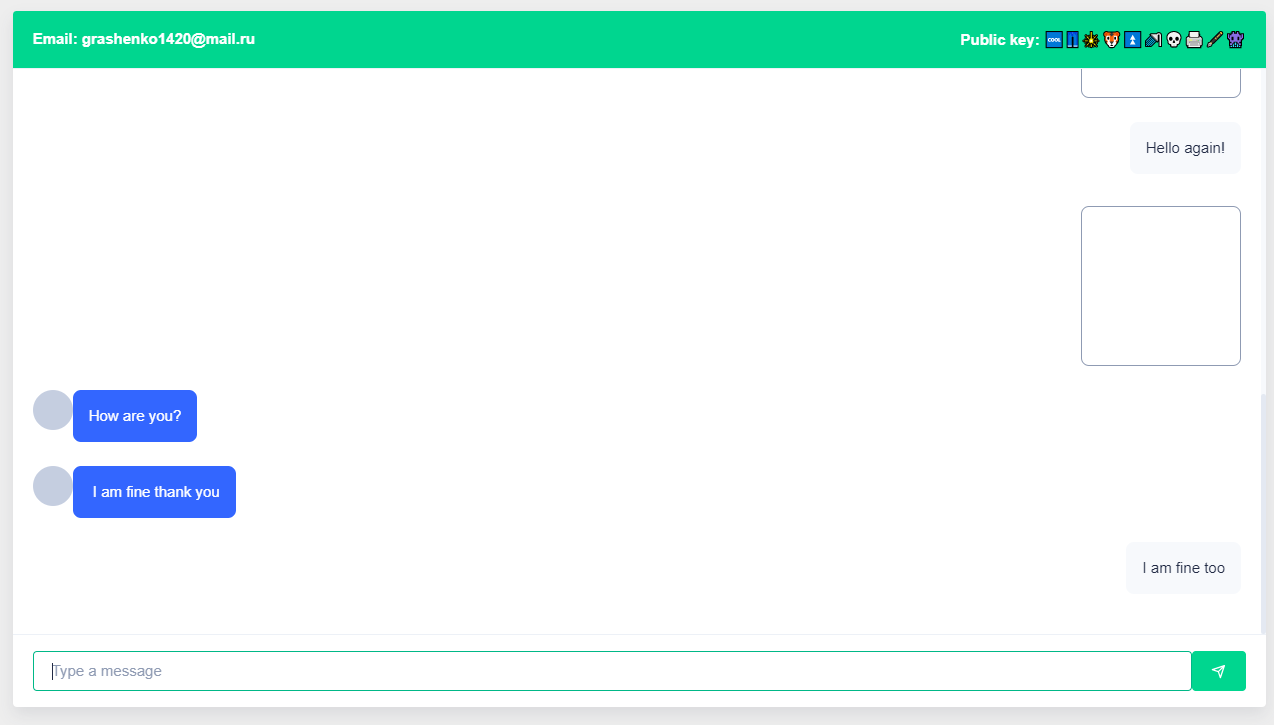


Рисунок 3.6 – Компонента *ChatComponent*

Следующей компонентой является *FriendsComponent*. Эта компонента служит для отображения списка пользователе и запросов в друзья. Кроме этого в функционал этой компоненты входит:

* выбор пользователя;
* импорт ключевой информации;
* экспорт ключевой информации;
* удаление пользователя;
* прием заявок в друзья;

Общий вид компоненты можно видеть на рисунке 3.7.

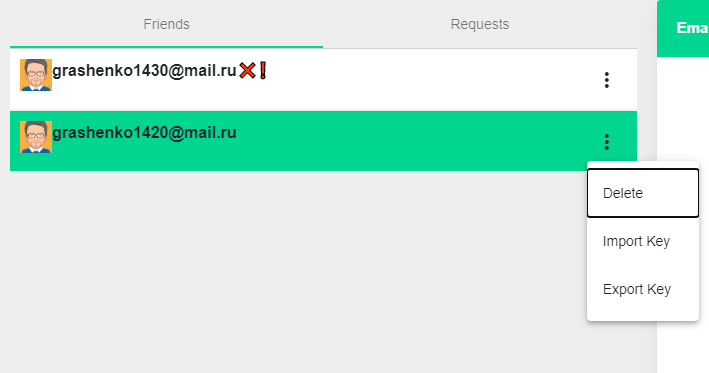


Рисунок 3.7 – Компонента *FriendsComponent*

Кроме вкладки *Friends* на компоненте присутствует вкладка *Requests*. На ней отображаются все заявки пользователей в друзья. На рисунке 3.8 можно видеть список заявок.

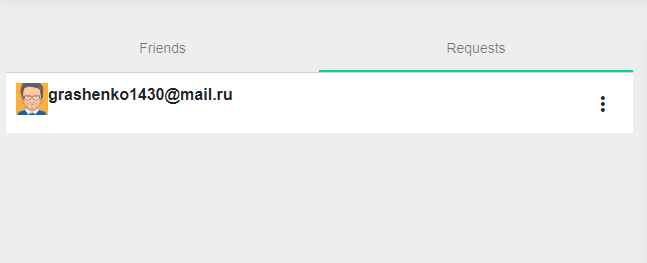


Рисунок 3.8 – Вкладка Requests

Также эта компонента синхронизирует локальное хранилище ключей с базой данных и отображает специальные символы возле имен пользователей в зависимости от текущего состояния синхронизации.

На листинге 3.9 представлена реализация функции *updateFriendsInfo*.

updateFriendsInfo() {

this.api.getFriendsRequests().subscribe((friendRequests) => {

this.friendsRequests = friendRequests;

this.api.getNotConfirmedRequests().subscribe((notConfirmedRequests) => {

this.сommon.parseNotConfirmedKeys(notConfirmedRequests).then(() => {

this.api.synchronizationRequests().subscribe((reqToSync)=> { this.сommon.synchronizationLocal(reqToSync);

})

this.api.getAllUsers().subscribe((resp) => {

resp.forEach((fr)=>{

const requestKeyLocal = JSON.parse(localStorage.getItem(`Request|${this.сommon.currentUser.userId}|${fr.userId}`));

const encryptKeyLocal = JSON.parse(localStorage.getItem(`Encrypt|${this.сommon.currentUser.userId}|${fr.userId}`));

fr.isLocalKeys = requestKeyLocal && encryptKeyLocal;

});

this.friends = resp;

});

});

});

});

});

}

Листинг 3.9 – Программная реализация *Lazy Loading*

Кроме пользовательских компоненты, были также использованы стандартные компоненты *ASP.NET Core*. Одной из таких компонент является *login.component.ts*.

private async processLoginCallback(): Promise<void> {

const url = window.location.href;

const result = await this.authorizeService.completeSignIn(url);

switch (result.status) {

case AuthenticationResultStatus.Redirect:

throw new Error('Should not redirect.');

case AuthenticationResultStatus.Success:

await this.navigateToReturnUrl(this.getReturnUrl(result.state));

break;

case AuthenticationResultStatus.Fail:

this.message.next(result.message);

break;

}

}

Листинг 3.10 – Программная реализация метода *processLoginCallback*

Компонента содержит в себе форму для заполнения логина и пароля, а также валидирует эти поля. Кроме этого, компонента имеет ссылки на страницы регистрации и отправки восстановления пароля на почту. Внешний вид данной компоненты можно видеть на рисунке 3.9.

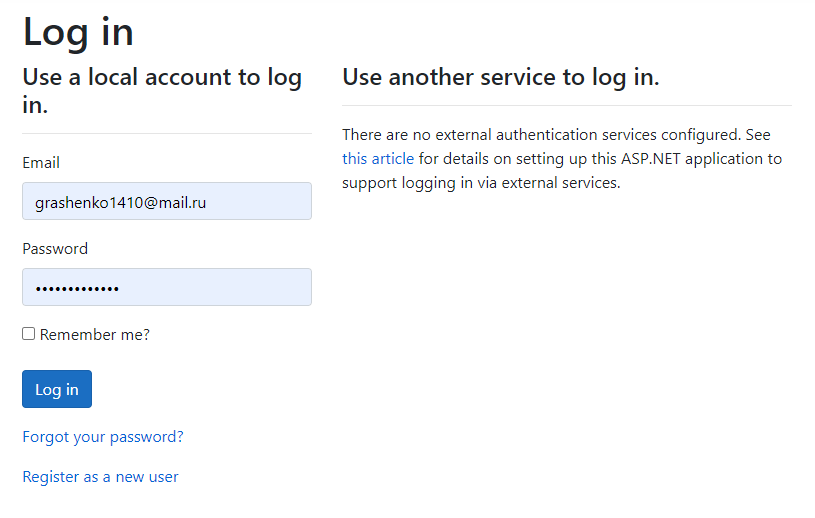


Рисунок 3.9 – Компонента *login.component.ts*

## Модели

Модели клиентского приложения *Angular* используются как для представления *JSON* объектов запросов и ответов сервера, так и внутри сервисов и компонентов *Angular*. Все модели находятся в папке *models*. На рисунке 3.10 можно видеть все файлы этой папки.

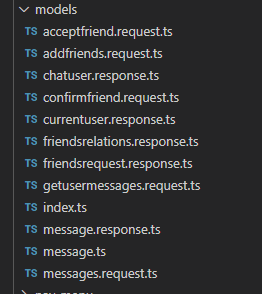


Рисунок 3.10 – Папка *models*

Каждая модель представляет собой класс *TypeScipt*. Поля моделей являются стандартными типами *TypeScipt*. Стандартные типы поддерживают полную совместимость с форматом *JSON*, что позволяет автоматически приводить данные с сервера к моделям клиентского приложения. На листинге 3.11 можно видеть реализацию модели *MessageResponse*.

export class MessageResponse {

public id: number;

public encryptedMessage: string;

public date: string;

public isReply: boolean;

public messageType: number;

}

Листинг 3.11 – Программная реализация модели *MessageResponse*

* 1. Websocket

Как говорилось выше, для использования протокола *wss* был установлен пакет *SignalR*. Данный протокол позволяет передавать сообщения без перезагрузки всей страницы. На стороне сервера *Websocket* представляет собой класс *Hub*, клиентские приложения подключаются к нему с помощью специального адреса. Этот адрес конфигурируется в классе *Startup.cs*. На листинге 3.12 можно видеть конфигурацию путей контроллера и хаба.

app.UseEndpoints(endpoints =>

{

endpoints.MapHub<ChatHub>("/chatHub");

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller}/{action=Index}/{id?}");

endpoints.MapRazorPages();

});

Листинг 3.12 – Конфигурация путей контроллера и хаба

После конфигурации клиентское приложение использует путь *chatHub*, для подключения к хабу. На клиентское приложение установлен пакет *@aspnet/signalr*. Данный пакет предоставляет класс *HubConnectionBuilder*, в классе содержатся два метода On и Invoke. С помощью метода On, добавляется обработчик вызова метода с сервера, а метод Invoke наоборот вызывает метод сервера и передает ему параметры. Название метода указывается как параметр, для метода на стороне сервера было выбрано название *Send* а на стороне клиента *Receive*. Кроме этого класс *HubConnectionBuilder* предоставляет возможность подключения к хабу используя пользовательские параметры. В качестве пользовательских параметров могу выступать как обычные данные запроса, так и заголовки запроса. При подключении к хабу клиентское приложение передает в качестве параметра токен аутентификации., который в последствии считывается сервером и на основе его можно идентифицировать пользователя, которые послал запрос.

На листинге 3.13 можно видеть пример подключения и реализации метода *On*.

this.hubConnection = new HubConnectionBuilder()

.withUrl("/chatHub", { accessTokenFactory: () => token })

.build();

this.hubConnection.start();

this.hubConnection.on('Receive', async (mes, type) => {

});

Листинг 3.13 – Подключение к хабу

* 1. Вывод по разделу

В результате выполнения дипломного проекта было разработано веб-приложение «Мессенджер со сквозным шифрованием». Все поставленные задачи перед проектом были выполнены. Основные сложности связанные с шфированием и харнением ключей на стороне клиента решились стандартными способами с минимальной потерей производительности приложения. Кроме этого был спроектировна сокет на основе бибилотеки *SignalR*.

Полученное веб-приложение не требует специфического программного обеспечения, для его запуска необходимо наличие установленного интернет-браузера на компьютере пользователя, доступ в сеть Internet любым доступным и удобным для пользователя способом, а также доступ к имеющимся данным. Проект является интуитивно понятным и простым. В программном средстве использованы устойчивые формы записи кода (использование обработчиков ошибок) благодаря чему повышается отказоустойчивость приложения.

Веб-ресурс справляется со всеми поставленными задачами и целями, а именно:

* спроектирована и разработана структура базы данных;
* разработан интерфейс мессенджера;
* разработан протокол обмена общим секретом;
* разработан формат хранения ключей;
* разработан сокет передачи сообщений;
* реализованы методы шифрования;
* реализована аутентификация и регистрация пользователей;
* реализована функция передачи файлов;
* реализован поиск по сообщениям на стороне клиента;
* разработан контроллер для обработки запросов клиентского приложения.

Спроектированная архитектура проекта, позволит добавлять и усовершенствовать функциональные возможности и поддерживать работу системы в будущем.

1. Тестирование приложения

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 04.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 4 Тестирование | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 5 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

Тестирование является неотъемлемой частью процесса разработки программного продукта, так как данный этап позволяет найти неисправности в работе приложения и исправить их.

В данной главе будут описаны некоторые тесты, проведенные для выявления и устранения неисправностей разработанного программного продукта.

Для начала представим названия тест-кейсов, которые и будем рассматривать ниже:

* отправка сообщения;
* импорт ключа;
* экспорт ключа;
* отправка файла;
* удаление пользователя;
  1. Отправка сообщения

Для отправки сообщения пользователю необходимо ввести сообщение в поле *Type a message*, после чего нажать клавишу *Enter*, либо на специальную кнопку справа от поля ввода. Сообщение может содержать любые символы и быть любой длины.

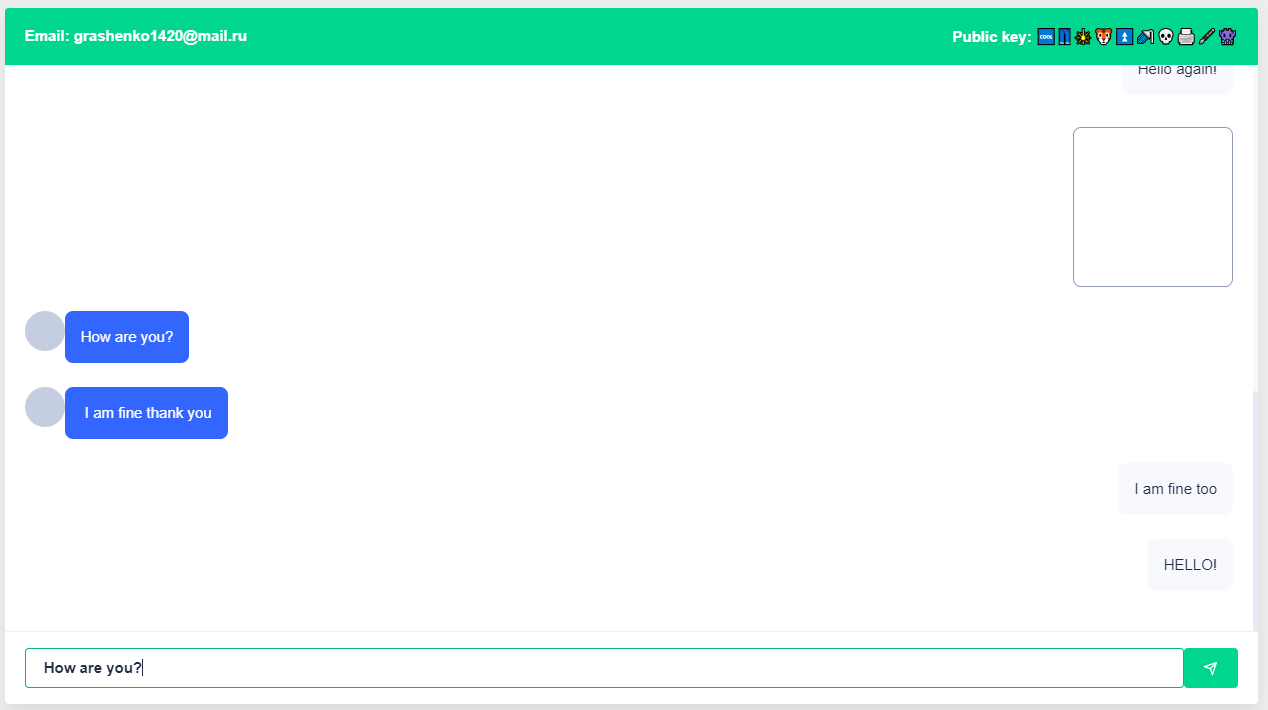


Рисунок 4.4 – Пример отправки сообщения

После отправки сообщения запустится анимация добавления сообщения, далее пользователь сможет увидеть свое сообщения в поле чата справа, на фоне серого цвета.

* 1. Импорт ключа

Импорт ключа необходим только тогда, когда локальное хранилище пользователя не синхронизировано с базой данных, это означает, что запись о ключе есть в базе данных, но в браузере ключ отсутствует.

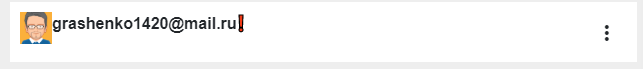


Рисунок 4.5 – Пользователь без ключа шифрования

Если пользователь видит такой символ, ему необходимо открыть контекстное меню и выбрать пункт *Import key*.

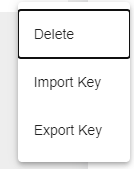


Рисунок 4.6 – Контекстное меню

На рисунке 4.6 можно видеть контекстное меню и пункт *Import Key*.

После выбора пункта *ImportKey* необходимо выбрать файл в формате *.data,* в котором содержатся ключ для данного пользователя. Далее программа предложит пользователю ввести пароль от файла.

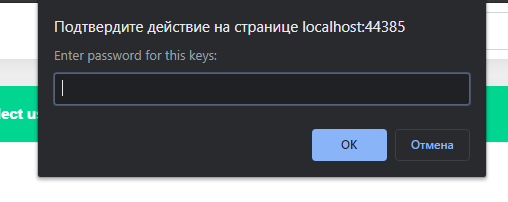


Рисунок 4.7 – Поле ввода пароля

В случае неверно введенного пароля будет выведено соответствующее сообщение.

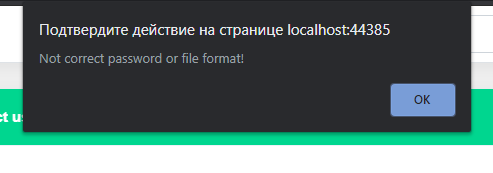


Рисунок 4.8 – Сообщение об ошибке пароля

Далее пользователь может повторить ввод пароля.

* 1. Экспорт ключа

Экспорт ключа доступен если у пользователя ключ присутствует в локальном хранилище. Доступ к данной функции, также доступен из контекстного меню в пункте *Export Key*. После чего пользователю будет предложено ввести пароль на файл.

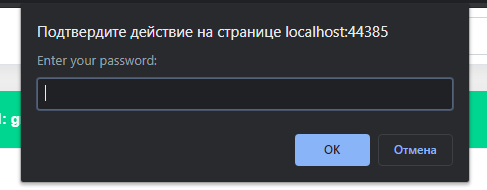


Рисунок 4.9 – Сообщение об ошибке пароля

Пароль не имеет ограничений по формату и по длине. После ввода пароля начнется скачивание файла на компьютер. Далее этот файл можно будет использовать для импорта ключей.

* 1. Отправка файла

Приложение поддерживает любые форматы форматируемы в строку base64. Большинство файлов изображений, видео, документов поддерживают данную конвертацию.

Для того чтобы отправить файл необходимо перетащить его в компоненту чата. Далее отпустить клавишу мыши, после чего выше поля ввода сообщения появится соответствующая иконка.

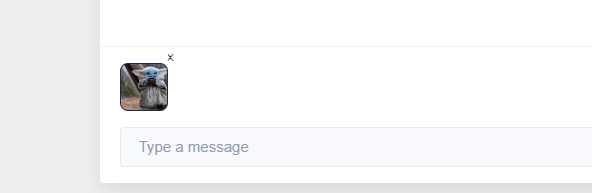


Рисунок 4.10 –Пример иконки файла

После этого можно отправлять файл. Для того чтобы открыть файл необходимо кликнуть на него. На рисунке 4.11 можно видеть пример строки base64 в браузере.

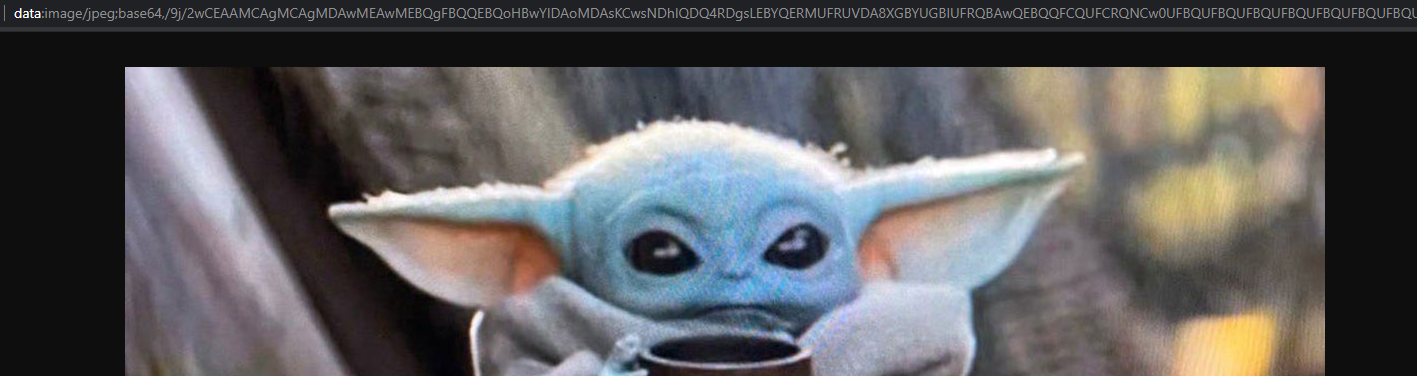


Рисунок 4.11 –Пример файла *base64*

* 1. Заявка в друзья

Перед началом чата пользователю необходимо отправить заявку в друзья, после подтверждения можно начать обмен сообщениями.

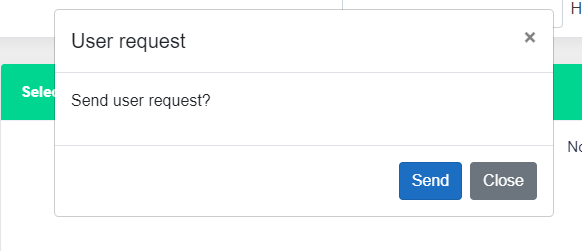


Рисунок 4.12 – Окно подтверждения запроса

При этой процедуре производится обмен общим секретом.

Далее пользователю, которому отправили запрос, придет уведомления в вкладку *Requests*, где он может подтвердить или отменить запрос.

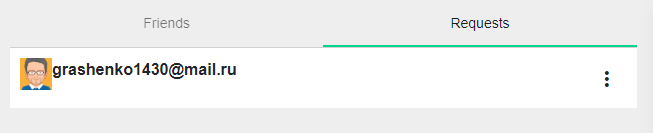


Рисунок 4.13 – Новый запрос в друзья

После нажатия на запрос пользователь может отменить либо принять заявку в друзья.

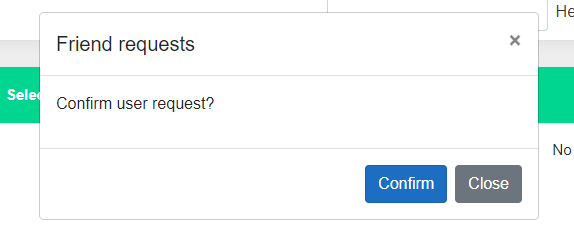


Рисунок 4.14 – Подтверждение запроса в друзья

В случае если пользователь подтвердит запрос, будет создан новый мессенджер.

* 1. Удаление пользователя

Для удаления пользователя в контекстном меню находится специальная вкладка *Delete*, на рисунке 4.15 можно видеть данную вкладку.

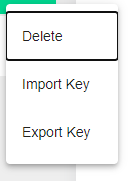


Рисунок 4.15 – Удаление пользователя

После удаления локальное хранилище будет очищено от ключей данного пользователя, все сообщения в базе данных будут удалены.

Вывод по разделу

В данной главе был выполнен контрольный пример для обнаружения ошибок в поведении системы и выполнен ряд тестов для проверки ее стабильности. В результате проделанной работы установлено, что система стабильно работает, все тесты были выполнены успешно, а именно:

* отправка сообщения;
* импорт ключа;
* экспорт ключа;
* отправка файла;
* удаление пользователя;

В ходе тестирования было выявлено, что интерфейс обладаете достаточной функциональностью и удобством, а дизайн приложение имеет благоприятный для пользования вид.

1. Руководство пользователя

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 05.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 5 Руководство пользователя | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 4 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

Системные требования

Системные требования проекта обусловлены наличием специфичных функций таких как сквозное шифрование. Стоит отметить что для работы необходим соединение интернет. Главное требования к браузерам – поддержка *WebCryptoApi*. На момент написания записки такую поддержку имеют:

* *Chrome 11*;
* *Edge 12*;
* *Firefox 26*;
* *Internet Explorer 11*;
* *Opera 15*;
* *Safari 6*.1;
* *Android webview*;
* *Chrome for Android 18*;
* *Firefox for Android 26*;
* *Opera for Android 14*;
* *Safari on iOS 6.1*;
* *Samsung Internet 1*.*0*.

Авторизация пользователя

Для проверки авторизации пользователя используются логин и пароль. Форма авторизации представлена на рисунке 5.1.

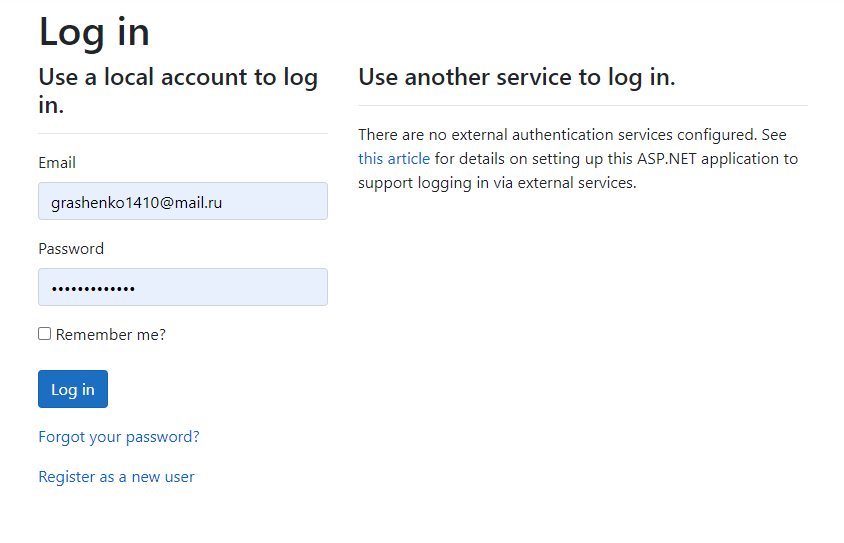


Рисунок 5.1 – Форма авторизации

При введении логина не имеющегося в базе данных, либо пароля не совпадающего с введенным логином выводится сообщение о неправильном логине либо пароле.

На рисунке 5.2 можно видеть, что при вводе неверного пароля или *email* выводится сообщения об ошибке.

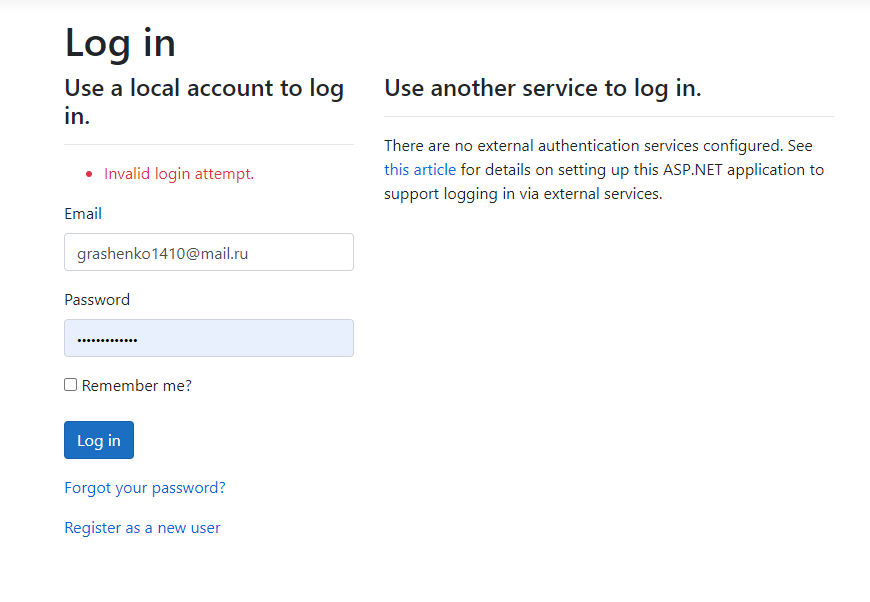
ы

Рисунок 5.2 – Сообщения о неверных данных

Регистрация пользователя

При регистрации пользователю необходимо ввести *Email*, пароль и подтверждение пароля. При этом валидируются все три поля. *Email* должен соответствовать стандартному виду. Пароль должен быть длиной от шести до ста символов, содержать числа, латинские буквы, иметь хотя бы одну заглавную букву и символ.

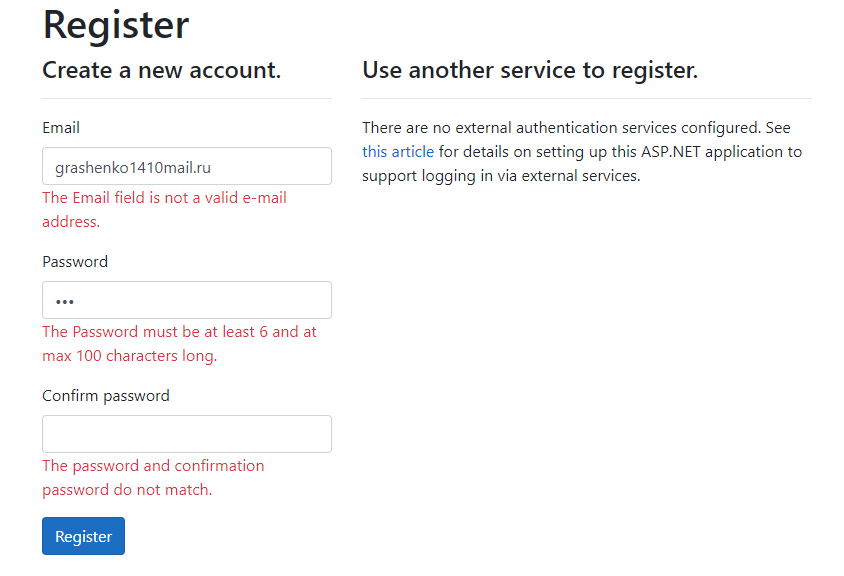


Рисунок 5.3 – Форма регистрации

После успешной регистрации пользователю необходимо зайти на страницу авторизации и ввести логин и пароль.

* 1. Добавление пользователя в друзья

Для добавления пользователя в список друзей необходимо кликнуть на него в списке друзей. После чего нажать *Confirm* в модальном окне. Далее, кода пользователь подтвердит вашу заявку, вам станет доступен чат с ним. На рисунке 5.4 можно видеть окно подтверждения заявки.

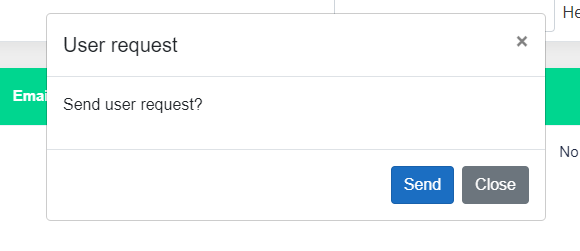


Рисунок 5.4 – Окно подтверждения отправки запроса в друзья

Кроме того, у пользователя не будут отображаться знаки рядом с именем в списке друзей. На рисунке 5.5 можно видеть пользователя готового к переписке.



Рисунок 5.5 – Окно подтверждения отправки запроса в друзья

При использовании приложения можно заметить несколько специальных знаков. На рисунке 5.6 можно видеть символы, обозначающие состояние пользователя.

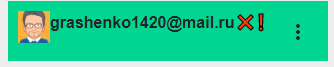


Рисунок 5.6 – Пользователь со специальными символами

Красный крест обозначает, что данный пользователь не является другом. Восклицательный знак обозначает, что в локальном хранилище отсутствуют ключи шифрования для переписки.

* 1. Поиск по сообщениям

Для поиска необходимо ввести в поле, расположенное в шапке, строку. Далее, по мере ввода, сообщения будут фильтроваться, в чате останутся лишь те сообщение, удовлетворяющие строке поиска.

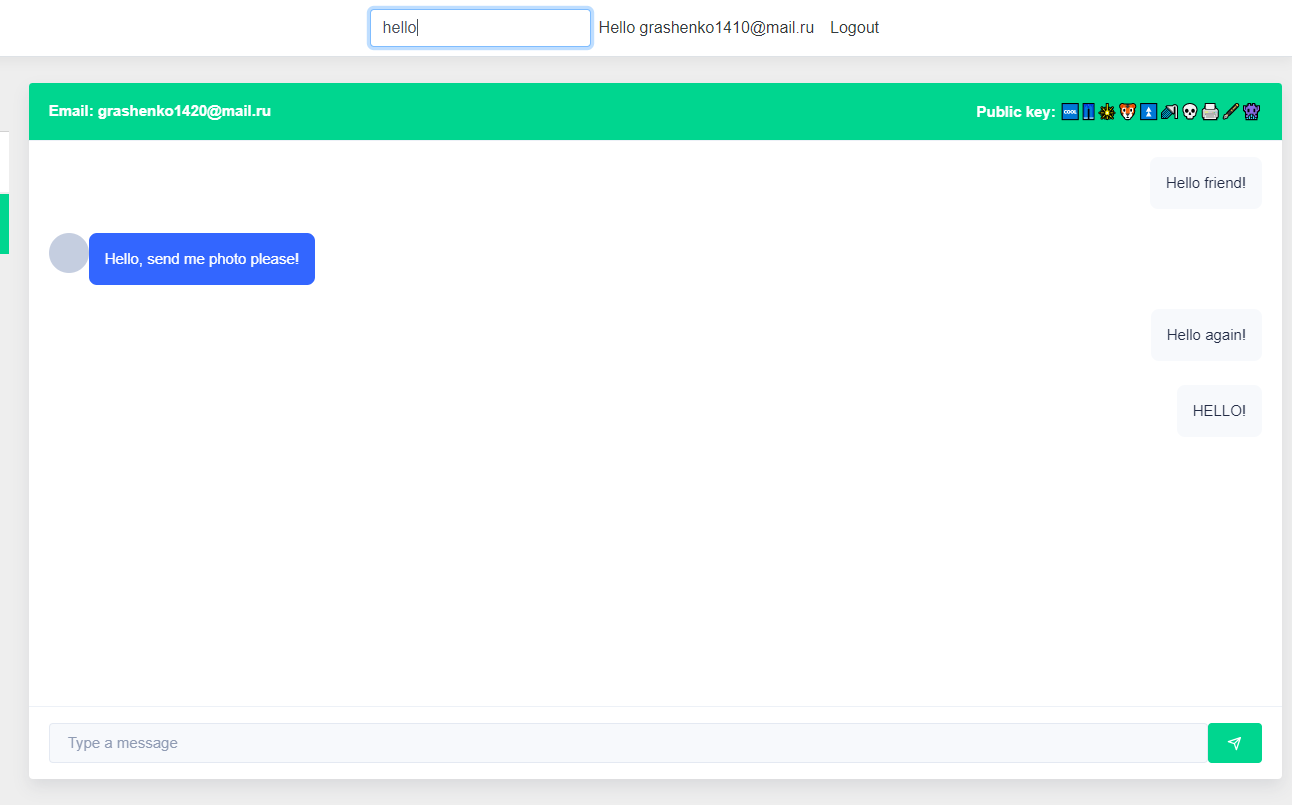


Рисунок 5.7 – Пример поиска по сообщениям

В поле ввода можно ввести строку любой длины, с любым количеством символов.

Вывод по разделу

Цель данного раздела заключалась в создании руководства пользователя, чтобы предоставить пользователю возможность самостоятельно решать свои прикладные задачи с помощью данного веб-приложения. Данное руководство служит кратким введением в предметную область и ознакомлением со всеми возможностями веб-приложения, а также содержит краткое описание конкретных процедур решения задач.

Данный раздел необходим для ознакомления пользователя с внешним видом, интерфейсом и функционалом веб-приложения. Данное руководство содержит описание следующих страниц:

* страница авторизация пользователя;
* страница регистрации;
* поиск по сообщениям.

То образом руководство пользователя полезно при возникновении вопросов касательно использования приложения и навигации между его страниц.

1. Экономическое обоснование цены программного средства

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 06.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | 6 Экономическое обоснование цены программного средства | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 9 |
| Консульт. | | Евлаш А.И. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

Главной целью экономического раздела является экономическое обоснование целесообразности разработки программного средства (ПС), представленного в дипломном проекте. В данном разделе пояснительной записки проводится расчет затрат на всех стадиях разработки и расчет экономии основных видов ресурсов в связи с использованием данного ПС.

Разработка проектов ПС требует разнообразных затрат и нередко значительных объемов ресурсов (трудовых, материальных, финансовых). В связи с этим, разработка и реализация каждого проекта должна быть обоснована, как технически, так и экономически.

Общая характеристика разрабатываемого программного средства

Программное средство создано при помощи среды разработки *Visual Studio* и языка программирования C#. Для реализации функционала использованы такие технологии, как .*NET*, *ASP.NET Core*, *Entity Framework Core*, *SignalR*. Также на стороне браузера был использован фреймворк *Angular*. В качестве базы данных использовалась база данных *MS SQL*. Разработанный программный продукт позволит выполнять следующие действия:

– регистрация и авторизация пользователей;

– возможность добавления пользователя в список контактов;

‒ удаление пользователя из списка контактов;

– обмен ключевой информации;

– пересылка сообщений без перезагрузки страницы;

– пересылка видео, документов, фотографий в зашифрованном виде;

– экспорт и импорт ключевой информации в файл;

– возможность установить пароль на файл при экспорте;

– поиск по сообщениям на стороне клиента;

– просмотр ключевой информации в виде набора *emoji*-смайлов;

Данный раздел служит для определения затрат, произведенных на всех стадиях разработки программного средства. ПО будет использовано для дальнейшей реализации на рынке.

* 1. Исходные данные и маркетинговый анализ

Источниками исходных данных для данных расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчёта

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 1 |
| Норматив дополнительной заработной платы | % | Ндз | 15 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34 |
| Коэффициент изменения скорости  обработки информации |  | Кск | 0,6 |
| Норма расхода материалов в расчете на сто строк исходного кода | руб. | Нм | 0,46 |
| Норматив машинного времени на  отладку ста строк исходного кода равен | машино-час | Нмв | 15 |
| Норматив прочих затрат | % | Нпз | 18 |
| Норматив накладных расходов | % | Нобп, обх | 152 |
| Норматив расходов на сопровождение и адаптацию | % | Нрса | 17 |
| Ставка НДС | % | Нндс | 20 |

В ходе маркетингового анализа было установлено следующее:

* средняя цена разработки программного продукта по теме дипломного проекта составляет 7300 рублей;
* средняя цена разработки дополнительных алгоритмов обработки ошибочных ситуаций в приложении составляет 2700 рублей;
* средняя цена разработки дизайна приложения составляет 1000 рублей.

В итоге полная стоимость разработки продукта составила 11000 рублей.

Данная цена является средней на рынке разработки программного обеспечения.

* 1. Методика обоснования цены

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершённые и имеющие товарный вид ПС, реализуемые покупателям по рыночным отпускным ценам. Все завершённые разработки ПС являются научно–технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчётом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счёт:

* снижения трудоёмкости расчётов и алгоритмизации программирования и отладки программ;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности в результате использования ПС.

Стоимостная оценка ПС у разработчиков предполагает определение затрат, что включает следующие статьи:

* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на оплату машинного времени;
* прочие прямые затраты;
* накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена ПС.

* 1. Объем программного средства

Для оценки объёма программного средства, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем программного средства *V*o, вычисляется как сумма объёмов *Vi* каждой из *n* его функций (формула 6.1).

|  |  |  |
| --- | --- | --- |
|  | , | (6.1) |

где *Vi* – объем функции программного средства;

*n* – общее число функций.

В таблице 6.2 представлены функции, присутствующие в рассматриваемом программном средстве и соответствующий им объем в условных машино–командах [15].

Таблица 6.2 – Содержание и объем функций в программном средстве

|  |  |  |
| --- | --- | --- |
| № функции | Содержание функции | Объем, условных машино-команд |
| 101 | Организация ввода информации | 520 |
| 102 | Контроль, предварительная обработка | 320 |
| 111 | Управление вводом/выводом | 800 |
| 202 | Взаимодействие между компонентами системы | 1600 |
| 401 | Взаимодействие с базой данных | 950 |
| 402 | Вспомогательные методы | 800 |
| 506 | Обработка ошибочных и сбойных ситуаций | 350 |
| 707 | Графический вывод результатов | 1200 |

Опираясь на данные таблицы 5.2, можно определить объем программного средства, разработанного в ходе дипломного проектирования:

*Vo* = 520 + 320 + 800 + 1600 + 950 + 800 + 350 + 1200 = 6 540 (машино–команд).

Уточнённый объем программного средства *V*o*/* вычисляется по формуле 6.2.

|  |  |  |
| --- | --- | --- |
|  | , | (6.2) |

где *V*o – объем программного средства, усл. машино-команд;

Кск - коэффициент изменения скорости обработки информации.

Исходя из вычисленного объёма программного средства, можно определить уточненный объем программного средства:

*V*o*/* = 6 540 ⋅ 0,6 = 3 924 (условных машино–команд).

* 1. Основная заработная плата

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере веб–программирования на ASP.Net. В итоге было установлено, что средняя месячная заработная плата на позиции junior составляет 900 рублей.

Проект разрабатывался одним человеком на протяжении двух месяцев. Таким образом, основная заработная плата будет рассчитываться по формуле (5.3):

|  |  |  |
| --- | --- | --- |
|  | , | (6.3) |

где Соз– основная заработная плата, руб.;

Траз – время раработки, месяцев;

Краз – количество разработчиков, человек;

Сзп – средняя месячная заработная плата.

|  |  |  |
| --- | --- | --- |
|  | руб. |  |

* 1. Дополнительная заработная плата

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле (6.4):

|  |  |  |
| --- | --- | --- |
|  | , | (6.4) |

где Соз– основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

руб.

* 1. Отчисления в Фонд социальной защиты населения

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 5.5:

|  |  |
| --- | --- |
|  | (6.5) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата на конкретное ПС, тыс. руб.;

Нфсзн– норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 5.6:

|  |  |
| --- | --- |
|  | (6.6) |

руб.

руб.

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 8,28 руб., а в фонд социальной защиты населения – 703,8 руб.

* 1. Расходы на материалы

Сумма расходов на материалы СМ определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода НМ на уточнённый объем программного средства *V*o*/* (формула 5.7).

|  |  |  |
| --- | --- | --- |
|  | , | (6.7) |

где НМ - норма расхода материалов в расчете на сто строк исходного кода;

*V*o*/* - уточнённый объем программного средства.

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб., можно определить сумму расходов на материалы:

СМ = 0,46 ⋅ 3 924 / 100 = 18,05 руб.

* 1. Расходы на оплату машинного времени

Сумма расходов на оплату машинного времени Смв определяется как произведение стоимости одного машино–часа Смч на уточнённый объем программного средства *V*o*/* и на норматив расхода машинного времени на отладку ста строк исходного кода НМВ (формула 5.8).

|  |  |  |
| --- | --- | --- |
|  | , | (6.8) |

где Смч - стоимость одного машино–часа;

*V*o*/* - уточнённый объем программного средства;

НМВ - норматив расхода машинного времени на отладку ста строк исходного кода.

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 15, можно определить сумму расходов на оплату машинного времени:

Смв = 0,06 ⋅ 3 924 ⋅ 15 / 100 = 35,32 руб.

* 1. Прочие прямые затраты

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз (формула 5.9).

|  |  |  |
| --- | --- | --- |
|  | , | (6.9) |

Спз = 1800 ⋅ 18 / 100 = 324 (руб.).

* 1. Накладные расходы

Сумма накладныхрасходовСобп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх (формула 5.10).

|  |  |  |
| --- | --- | --- |
|  | , | (6.10) |

где Соз - основная заработная плата исполнителей на конкретное программное средство;

Нобп,обх - норматив накладных расходов в целом по организации.

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

Собп,обх = ⋅ 152 / 100 = руб.

* 1. Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов (формула 6.11).

|  |  |  |
| --- | --- | --- |
|  | Ср = Соз + Сдз + Сфсзн + Сбгс + См + Смв + Спз + Собп,обх , | (6.11) |

где Ср – сумма расходов, руб.;

Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата, руб.;

Сфсзн – сумма отчислений в Фонд социальной защиты населения, руб.;

См – сумма расходов на материалы, руб.;

Смв – сумма расходов на оплату машинного времени, руб.;

Спз – сумма прочих затрат, руб.;

Собп,обх – сумма накладных расходов, руб.

Ср = + + + 8,28 + 18,05 + 35,32 + 324 + = 5895,45 руб.

* 1. Расходы на сопровождение и адаптацию

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса (формула 6.12).

|  |  |  |
| --- | --- | --- |
|  | , | (6.12) |

где Ср - сумма расходов на разработки;

Нрса - норматив расходов на сопровождение и адаптацию.

Срса = 5895,45 ⋅ 17 / 100 = 1002,23 (руб.).

* 1. Полная себестоимость

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса (формула 6.13).

|  |  |  |
| --- | --- | --- |
|  |  | (6.13) |

Сп = 5895,45 + 1002,23 = 6897,68 (руб.).

* 1. Определение цены, оценка эффективности

При рассмотрении аналогов было установлено, что данные продукты являются коммерческими и найти стоимость аналогичных продуктов найти сложно. В своём представлении данный продукт представляет веб–приложение, поэтому в качестве среднее рыночной цены можно взять среднюю цену разработки веб–ресурсов, которая равна 11000 руб.

Прибыль рассчитывается по формуле (6.14):

|  |  |  |
| --- | --- | --- |
|  | , | (6.14) |

где Ппс – прибыль от реализации программного средства, руб.;

Цр – средняя рыночная цена продукта, руб.;

Сп – полная себестоимость программного средства, руб.

Ппс = 11000 / 1,2 – 6897,68 = 2268,98 (руб.).

Уровень рентабельности разработанного программного средства определяется по формуле (6.15):

|  |  |  |
| --- | --- | --- |
|  | , | (6.15) |

где Урент – уровень рентабельности программного средства, %

Сп – полная себестоимость программного средства, руб.;

Ппс – прибыль от реализации программного средства, руб.

Ур = 2268,98 / 6897,68 \* 100 = 32,89 (%.).

* 1. Вывод по разделу

В таблице 6.3 представлены результаты расчётов для основных показателей данной главы в краткой форме.

Таблица 6.3 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 2 |
| Количество программистов, чел. | 1 |
| Зарплата с отчислениями, руб. | 2 782,08 |
| Расходы на материалы, оплату машинного времени, прочие, руб | 377,37 |
| Накладные расходы, руб | 2 736 |
| Себестоимость разработки программного средства, руб. | 5 895,45 |
| Расходы на сопровождение и адаптацию, руб. | 1 002,23 |
| Полная себестоимость, руб. | 6 897,68 |
| Цена аналога, руб. | 11 000 |
| Прибыль от реализации, руб. | 2 268,98 |
| Рентабельность разработки, % | 32,89 |

Разработка программного средства, осуществляемая одним программистом в течении двух месяцев, при заданных условиях обойдется компании в 6897,68 руб. Реализации данного программного средства по среднерыночной цене в 11000 руб., принесет прибыль компании в сумме 2268,98 рублей. Уровень рентабельности составит 32,89 %.

Данное веб-приложение обеспечивает шифрование на стороне клиента в различных браузерах, что позволяет клиентам передавать сообщения, файлы, фото, видео не опасаясь, что посредник сможет их перехватить. Кроме этого, у них нет необходимости доверять сторонним центрам сертификации или серверам поставщика, каждый клиент может убедиться в сохранности и подлинности передаваемых сообщений. В современном мире и бизнесе эта функция становится все более и более востребована, поэтому данное приложение может составить другим продуктам достойную конкуренцию.

Заключение

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 00.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | Заключение | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 1 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

В ходе выполнения дипломного проекта были рассмотрены основные аналоги, были рассмотрены их преимущества и недостатки, изучены возможности рассматриваемых программных средств. Произведен обзор теоретического материала по теме дипломного проекта. Также был выполнен патентный поиск аналогов разрабатываемого проекта.

Были проанализированы и выбраны основные технологии и средства для разработки дипломного проекта. Программное средство реализовано на языке *C#* на платформе *.Net Core*. В качестве системы управления базой данных использована СУБД *Microsoft SQL Server*.

При разработке программного средства была спроектирована структура базы данных, а также была модифицирована файловая структура проекта, соответствующая паттерну проектирования *MVC*. В качестве клиентской части, было разработано приложение *Angular*.

При составлении руководства пользователя была подробно описана работа с программным средством. Так же было проведено тестирование, которое показало, что разработанное программное средство соответствует заданным требованиям.

При рассмотрении технико-экономического обоснования разработанного проекта было рассчитано количество денежных затрат и трудозатрат на разработку программного средства, прибыль от реализации разработанного программного продукта.

Целью данного дипломного проекта являлась проектирование и разработка веб-приложения «Мессенджер со сквозным шифрованием». Для достижения поставленной цели были выполнены следующие задачи:

– проектирование и разработка структуры базы данных;

– разработка интерфейса веб-приложения;

– реализация алгоритма обмена ключевой информации;

– реализация импорта и экспорта ключевой информации;

– реализация загрузки сообщений по мере прокрутки страницы;

– проектирование формата хранения ключевой информации в браузере и в файлах для импорта;

– реализация авторизации и аутентификации на основе *ASP.NET CORE Identity*;

– разработка веб-сокета для отправки сообщений;

В рамках дипломного проекта было разработано программное средство, которое соответствует предъявленным требованиям и поставленным задачам.

**Список использованных источников**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 00.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | | Гращенко А.С. |  |  | Список использованных источников | Лит. | | | Лист | Листов |
| Пров. | | Нистюк О.А. |  |  |  | У |  | 1 | 1 |
| Консульт. | | Нистюк О.А. |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | | Рыжанкова А.С. |  |  |
| Утв. | | Смелов В.В. |  |  |

1. C# Reference [Электронный ресурс] / docs.microsoft.com – Режим доступа: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/> – Дата доступа: 14.03.2020.
2. Angular docs [Электронный ресурс] / angular.io – Режим доступа: https:// angular.io/docs – Дата доступа: 15.03.2020.
3. Identity [Электронный ресурс] / docs.microsoft.com – Режим доступа: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication> – Дата доступа: 29.03.2020.
4. SignlaR [Электронный ресурс] / npmjs.com – Режим доступа: <https://www.npmjs.com/package/@aspnet/signalr> – Дата доступа: 25.04.2020.
5. ES6Features [Электронный ресурс] / es6-features.org – Режим доступа: <http://es6-features.org> – Дата доступа: 22.04.2020.
6. SQL Server [Электронный ресурс] / docs.microsoft.com – Режим доступа: <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15> – Дата доступа: 27.03.2020.
7. Патентный поиск [Электронный ресурс] – Режим доступа: http://it4b.icsti.su/itb/ps/ps\_all.html. Дата доступа: 21.04.2020.
8. Патент [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%82%D0%B5%D0%BD%D1%82/. Дата доступа: 28.04.2020.
9. Рихтер, Дж. CLR via С#. Программирование на платформе Microsoft .NET Framework 4.0 / Дж. Рихтер. – 3-е изд. ¬– СПб.: Питер, 2012. – 928 с.
10. Руководство по ASP.NET Core 3 [Электронный ресурс] / <https://metanit.com/sharp/aspnet5/> – Дата доступа: 20.05.2020.
11. Web technology for developers Web Crypto API [Электронный ресурс] / <https://developer.mozilla.org> – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API> – Дата доступа: 15.05.2020.
12. Руководство по Angular 9 [Электронный ресурс] / https://metanit.com/web/angular2 – Дата доступа: 16.05.2020.
13. Google Patents [Электронный ресурс] – Режим доступа: https://patents.google.com/. Дата доступа: 29.04.2020.
14. Angular Material Guide [Электронный ресурс] / <https://material.angular.io/> – Режим доступа: <https://material.angular.io/guides> – Дата доступа: 09.05.2020.

ПРИЛОЖЕНИЕ A

Контроллер ChatController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ChatServer.ChatModels;

using ChatServer.ChatData;

using ChatServer.Requests;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using ChatServer.Models;

using ChatServer.Reponses;

using System.Security.Claims;

using Microsoft.AspNetCore.SignalR;

using ChatServer.SignalR;

namespace ChatServer.Controllers

{

[Authorize]

[ApiController]

[Route("[controller]")]

public class ChatController : ControllerBase

{

IHubContext<ChatHub> HubContext { get; set; }

public ChatController(IHubContext<ChatHub> hubcontext)

{

this.HubContext = hubcontext;

}

[HttpGet("GetFriendsRequests")]

public IEnumerable<FriendsRequestsVM> GetFriendsRequests()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var aspNetUsers = db.AspNetUsers.ToList();

var users = db.FriendsRequests.ToList().Where((re) => re.FriendId == userId && re.Type == 0 && re.IsConfirmed == false).Select((fr) => new FriendsRequestsVM { Id = fr.Id, RequestUserId = fr.RequestUserId, RequestUserName = aspNetUsers.FirstOrDefault((us) => us.Id == fr.RequestUserId)?.UserName, EncryptionKey = fr.EncryptionKey, RequestPublicKey = fr.RequestPublicKey, FriendId = fr.FriendId, Type = fr.Type, IsConfirmed = fr.IsConfirmed });

return users;

}

}

[HttpGet("GetNotConfirmedRequests")]

public IEnumerable<FriendsRequestsVM> GetNotConfirmedRequests()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var aspNetUsers = db.AspNetUsers.ToList();

var users = db.FriendsRequests.ToList().Where((re) => re.FriendId == userId && re.Type == 1 && re.IsConfirmed == false).Select((fr) => new FriendsRequestsVM { Id = fr.Id, RequestUserId = fr.RequestUserId, RequestUserName = aspNetUsers.FirstOrDefault((us) => us.Id == fr.RequestUserId)?.UserName, EncryptionKey = fr.EncryptionKey, RequestPublicKey = fr.RequestPublicKey, FriendId = fr.FriendId, Type = fr.Type, IsConfirmed = fr.IsConfirmed });

return users;

}

}

[HttpPost("AddFriendsRequests")]

public FriendsRequestsVM AddFriendsRequests(AddFriendsRequest request)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

if (db.FriendsRequests.ToList().Where((re) => re.FriendId == request.FriendId && re.RequestUserId == userId).FirstOrDefault() == null)

{

var fr = db.FriendsRequests.Add(new FriendsRequests { RequestUserId = userId, FriendId = request.FriendId, RequestPublicKey = request.PublicKey, Type = 0, IsConfirmed = false }).Entity;

var newRequest = new FriendsRequestsVM { Id = fr.Id, RequestUserId = fr.RequestUserId, EncryptionKey = fr.EncryptionKey, RequestPublicKey = fr.RequestPublicKey, FriendId = fr.FriendId, Type = fr.Type, IsConfirmed = fr.IsConfirmed };

db.SaveChanges();

return newRequest;

}

else

{

return null;

}

}

}

[HttpPost("ConfirmFriendRequest")]

public FriendsRequestsVM ConfirmFriendRequest(ConfirmFriendRequest request)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var req = db.FriendsRequests.Where((re) => re.FriendId == userId && re.RequestUserId == request.FriendId && re.Type == 0).FirstOrDefault();

req.IsConfirmed = true;

var fr = db.FriendsRequests.Add(new FriendsRequests { RequestUserId = userId, FriendId = request.FriendId, EncryptionKey = request.EncryptedKey, RequestPublicKey = null, Type = 1, IsConfirmed = false }).Entity;

var newRequest = new FriendsRequestsVM { Id = fr.Id, RequestUserId = fr.RequestUserId, EncryptionKey = fr.EncryptionKey, RequestPublicKey = fr.RequestPublicKey, FriendId = fr.FriendId, Type = fr.Type, IsConfirmed = fr.IsConfirmed };

db.SaveChanges();

return newRequest;

}

}

[HttpPost("AcceptFriendRequest")]

public bool AcceptFriendRequest(AcceptFriendRequest request)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var fr = db.FriendsRequests.Where((re) => re.FriendId == userId && re.RequestUserId == request.FriendId && re.Type == 1).FirstOrDefault();

fr.IsConfirmed = true;

db.FriendsRelations.Add(new FriendsRelations { UserId = userId, FriendId = request.FriendId });

db.FriendsRelations.Add(new FriendsRelations { UserId = request.FriendId, FriendId = userId });

db.SaveChanges();

return true;

}

}

[HttpGet("GetFriends")]

public IEnumerable<FriendsRelationsVM> GetFriends()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var req = db.FriendsRelations.Where((fr) => fr.UserId == userId).ToList().Select((fr) => new FriendsRelationsVM { Id = fr.Id, FriendId = fr.FriendId, UserId = fr.UserId });

return req;

}

}

[HttpGet("GetAllUsers")]

public IEnumerable<ChatUserVM> GetAllUsers()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var relations = db.FriendsRelations.ToList();

var users = db.AspNetUsers.ToList().Where((u) => u.Id != userId).Select((u) => new ChatUserVM { UserId = u.Id, UserName = u.UserName, IsFriend = relations.Where((fr) => fr.UserId == userId && fr.FriendId == u.Id).FirstOrDefault() == null ? false : true });

return users;

}

}

[HttpGet("GetCurrentUser")]

public CurrentUserVM GetCurrentUser()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

if (userId != null)

{

using (ChatContext db = new ChatContext())

{

var user = db.AspNetUsers.FirstOrDefault((u) => u.Id == userId);

var userResponse = new CurrentUserVM() { UserId = user.Id, UserName = user.UserName };

return userResponse;

}

}

return null;

}

[HttpGet("SynchronizationRequests")]

public IEnumerable<FriendsRequestsVM> SynchronizationRequests()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var aspNetUsers = db.AspNetUsers.ToList();

var synchroRequests = db.FriendsRequests.ToList().Where((fr) => (fr.RequestUserId == userId || fr.FriendId == userId)).Select((fr) => new FriendsRequestsVM { Id = fr.Id, RequestUserId = fr.RequestUserId, RequestUserName = aspNetUsers.FirstOrDefault((us) => us.Id == fr.RequestUserId)?.UserName, EncryptionKey = fr.EncryptionKey, RequestPublicKey = fr.RequestPublicKey, FriendId = fr.FriendId, Type = fr.Type, IsConfirmed = fr.IsConfirmed });

return synchroRequests;

}

}

//[HttpPost("DeleteNotSynchroRequests")]

//public bool DeleteNotSynchroRequests(IEnumerable<FriendsRequestsVM> requests)

//{

// try

// {

// var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

// using (ChatContext db = new ChatContext())

// {

// requests.ToList().ForEach((req) =>

// {

// var itemsToRemove = db.FriendsRequests.Where((fr) => fr.Id == req.Id).ToList();

// itemsToRemove.ForEach((itemToRemove) =>

// {

// if (itemToRemove != null)

// {

// db.FriendsRequests.Remove(itemToRemove);

// }

// if (itemToRemove != null && itemToRemove.Type == 1)

// {

// var friendsToDelete = db.FriendsRelations.ToList().Where((fr) => (fr.UserId == itemToRemove.RequestUserId && fr.FriendId == itemToRemove.FriendId) || (fr.UserId == itemToRemove.FriendId && fr.FriendId == itemToRemove.RequestUserId));

// db.FriendsRelations.RemoveRange(friendsToDelete);

// }

// });

// });

// db.SaveChanges();

// return true;

// }

// }

// catch (Exception)

// {

// return false;

// }

//}

[HttpPost("GetUserMessages")]

public IEnumerable<MessageVM> GetUserMessages(GetUserMessagesRequest request)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

IEnumerable<MessageVM> messages = null;

using (ChatContext db = new ChatContext())

{

if (request.Id != null)

{

messages = db.UserMessages.Where((m) => ((m.IdTo == userId && m.IdFrom == request.FriendId)

|| (m.IdTo == request.FriendId && m.IdFrom == userId))

&& (m.Id < request.Id)).OrderByDescending((um) => um.Id).Take(10).ToList().Select((mes) => new MessageVM { Id = mes.Id, EncryptedMessage = mes.Message, IsReply = (mes.IdFrom == userId), Date = mes.Date, MessageType = mes.MessageType });

}

else

{

messages = db.UserMessages.Where((m) => (m.IdTo == userId && m.IdFrom == request.FriendId)

|| (m.IdTo == request.FriendId && m.IdFrom == userId)).OrderByDescending((um)=> um.Id).Take(10).ToList().Select((mes) => new MessageVM {Id = mes.Id, EncryptedMessage = mes.Message, IsReply = (mes.IdFrom == userId), Date = mes.Date, MessageType = mes.MessageType });

}

return messages;

}

}

[HttpPost("SendUserMessage")]

public int SendUserMessage(MessageRequest request)

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

using (ChatContext db = new ChatContext())

{

var message = new UserMessages() { IdFrom = userId, IdTo = request.FriendId, Message = request.Message, Date = DateTime.Now, MessageType = request.MessageType };

db.UserMessages.Add(message);

db.SaveChanges();

return message.Id;

}

}}}

ПРИЛОЖЕНИЕ Б

Компонента friends.component.html

<div \*ngIf="IsAuth">

<mat-tab-group mat-align-tabs="center">

<mat-tab label="Friends">

<div class="m-b-2 friend-container" \*ngFor="let friend of friends" (click)="selectFriend(friend)">

<div class="p-10 bg-white" [ngClass]="{'bg-green':friend.isSelected}">

<div class="media media-xs overflow-visible">

<a class="media-left" href="javascript:;">

<img src="https://bootdey.com/img/Content/avatar/avatar1.png" alt="" class="media-object img-circle">

</a>

<div class="media-body valign-middle">

<b class="text-inverse">{{friend.userName}}</b>

<label \*ngIf="!friend.isFriend">❌</label>

<label \*ngIf="!friend.isLocalKeys">❗</label>

</div>

<button mat-icon-button [matMenuTriggerFor]="menu" aria-label="Example icon-button with a menu"

(click)="selectContextMenu(friend, $event)">

<mat-icon>more\_vert</mat-icon>

</button>

<mat-menu #menu="matMenu">

<button mat-menu-item>Delete</button>

<button mat-menu-item><input type="file" id="import\_key" class="import-key" name="import\_key" (change)="importKey($event)" /> <label for="import\_key">Import Key</label></button>

<button mat-menu-item (click)=exportKey() >Export Key</button>

</mat-menu>

</div>

</div>

</div>

</mat-tab>

<mat-tab label="Requests">

<div class="m-b-2 friend-container" \*ngFor="let request of friendsRequests" (click)="selectRequest(request)">

<div class="p-10 bg-white">

<div class="media media-xs overflow-visible">

<a class="media-left" href="javascript:;">

<img src="https://bootdey.com/img/Content/avatar/avatar1.png" alt="" class="media-object img-circle">

</a>

<div class="media-body valign-middle">

<b class="text-inverse">{{request.requestUserName}}</b>

</div>

<button mat-icon-button [matMenuTriggerFor]="menu" aria-label="Example icon-button with a menu"

(click)="selectContextMenu(friend, $event)">

<mat-icon>more\_vert</mat-icon>

</button>

<mat-menu #menu="matMenu">

<button mat-menu-item>Delete</button>

</mat-menu>

</div>

</div>

</div>

</mat-tab>

</mat-tab-group>

<div \*ngIf="isShowModalSendRequest" class="modal">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">User request</h5>

<button type="button" class="close" (click)="deleteUserRequest()">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Send user request?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" (click)="sendUserRequest()">Send</button>

<button type="button" class="btn btn-secondary" data-dismiss="modal"

(click)="deleteUserRequest()">Close</button>

</div>

</div>

</div>

</div>

<div \*ngIf="isShowModalConfirmRequest" class="modal">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Friend requests</h5>

<button type="button" class="close" (click)="deleteConfirmRequest()()">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Confirm user request?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" (click)="sendConfirmRequest()">Confirm</button>

<button type="button" class="btn btn-secondary" data-dismiss="modal"

(click)="deleteConfirmRequest()">Close</button>

</div>

</div>

</div>

</div>

</div>

<div \*ngIf="!IsAuth">

Please login first!

</div>

ПРИЛОЖЕНИЕ В

Сервис crypto.service.ts

import { Injectable } from '@angular/core';

import { Observable, Subject } from 'rxjs';

const emojis = require('src/assets/emojis.json');

@Injectable()

export class CryptoService {

constructor() {

}

//RSA

public async generateRSAKeys() {

return await window.crypto.subtle.generateKey(

{

name: "RSA-OAEP",

modulusLength: 2048,

publicExponent: new Uint8Array([0x01, 0x00, 0x01]),

hash: { name: "SHA-256" },

},

true,

["encrypt", "decrypt"]

)

}

public async exportRSAKey(key: CryptoKey) {

return await window.crypto.subtle.exportKey(

"jwk",

key

);

}

public async importRSAPublicKey(key: JsonWebKey) {

return await window.crypto.subtle.importKey(

"jwk",

{

kty: key.kty,

e: key.e,

n: key.n,

alg: key.alg,

ext: key.ext,

},

{

name: "RSA-OAEP",

hash: { name: "SHA-256" },

},

true,

["encrypt"]

);

}

public async importRSAPrivateKey(key: JsonWebKey) {

return await window.crypto.subtle.importKey(

"jwk",

{

kty: key.kty,

e: key.e,

d: key.d,

dp: key.dp,

dq: key.dq,

n: key.n,

p: key.p,

q: key.q,

qi: key.qi,

alg: key.alg,

ext: true,

},

{

name: "RSA-OAEP",

hash: { name: "SHA-256" },

},

true,

["decrypt"]

);

}

public async encryptRSA(data: Uint8Array, publicKey: CryptoKey) {

return await window.crypto.subtle.encrypt(

{

name: "RSA-OAEP"

},

publicKey,

data

)

}

public async decryptRSA(data: Uint8Array, privateKey: CryptoKey) {

return await window.crypto.subtle.decrypt(

{

name: "RSA-OAEP"

},

privateKey,

data

)

}

//AES

public async generateAESKey() {

return await window.crypto.subtle.generateKey(

{

name: "AES-CTR",

length: 256,

},

true,

["encrypt", "decrypt"]

);

}

public async exportAESKey(key: CryptoKey) {

return await window.crypto.subtle.exportKey(

"jwk",

key

);

}

public async importAESKey(key: JsonWebKey) {

return await window.crypto.subtle.importKey(

"jwk",

{

kty: key.kty,

k: key.k,

alg: key.alg,

ext: key.ext,

},

{

name: "AES-CTR",

hash: { name: "SHA-256" }

},

true,

["encrypt", "decrypt"]

);

}

public async encryptAES(data: Uint8Array, key: CryptoKey) {

return await window.crypto.subtle.encrypt(

{

name: "AES-CTR",

counter: new Uint8Array(16),

length: 128,

},

key,

data

);

}

public async decryptAES(data: Uint8Array, key: CryptoKey) {

return await window.crypto.subtle.decrypt(

{

name: "AES-CTR",

counter: new Uint8Array(16),

length: 128

},

key,

data

);

}

public async getEmojiFromString(inputString: string, hashLength: number = 1) {

const hash = await window.crypto.subtle.digest('SHA-256', this.stringToArray(inputString));

const hexHash = this.buf2hex(hash);

const decimalHash = parseInt(hexHash, 16);

let emojiIndex = decimalHash % Math.pow(emojis.length, hashLength);

let emojiString = '';

for (let ii = 0; ii < hashLength; ii++) {

emojiString = `${emojis[emojiIndex % emojis.length]}${emojiString}`;

emojiIndex = Math.floor(emojiIndex / emojis.length);

}

return emojiString;

}

public stringToArray(bufferString) {

let uint8Array = new TextEncoder().encode(bufferString);

return uint8Array;

}

public arrayToString(bufferValue) {

return new TextDecoder().decode(bufferValue);

}

public arrayBufferTobase64(buffer: Uint8Array) {

var binary = '';

var len = buffer.byteLength;

for (var i = 0; i < len; i++) {

binary += String.fromCharCode(buffer[ i ]);

}

return window.btoa(binary);

}

public base64ToArrayBuffer(base64: string) {

var binary\_string = window.atob(base64);

var len = binary\_string.length;

var bytes = new Uint8Array(len);

for (var i = 0; i < len; i++) {

bytes[i] = binary\_string.charCodeAt(i);

}

return bytes;

}

public buf2hex(buffer: ArrayBuffer): string {

return Array.prototype.map.call(new Uint8Array(buffer), x => ('00' + x.toString(16)).slice(-2)).join('');

}

//Save with password

public getContentEncoding(message: string) {

const enc = new TextEncoder();

return enc.encode(message);

}

public getKeyMaterial(password: string) {

let enc = new TextEncoder();

return window.crypto.subtle.importKey(

"raw",

enc.encode(password),

"PBKDF2" ,

false,

["deriveBits", "deriveKey"]

);

}

public async setPassword(content: string, password: string) {

let keyMaterial = await this.getKeyMaterial(password);

let key = await this.getKey(keyMaterial);

let encoded = this.getContentEncoding(content);

return await window.crypto.subtle.encrypt(

{

name: "AES-GCM",

iv: new Uint8Array(12)

},

key,

encoded

);

}

public getKey(keyMaterial) {

return window.crypto.subtle.deriveKey(

{

"name": "PBKDF2",

salt: new Uint8Array(16),

"iterations": 100000,

"hash": "SHA-256"

},

keyMaterial,

{ "name": "AES-GCM", "length": 256 },

true,

["encrypt", "decrypt"]

);

}

public async resolvePassword(content: ArrayBuffer, password: string) {

let keyMaterial = await this.getKeyMaterial(password);

let key = await this.getKey(keyMaterial);

return await window.crypto.subtle.decrypt(

{

name: "AES-GCM",

iv: new Uint8Array(12)

},

key,

content

);

}

}

ПРИЛОЖЕНИЕ Г

Сервис api.service.ts

import { Injectable, Inject } from '@angular/core';

import { Observable, Subject } from 'rxjs';

import { HttpClient } from '@angular/common/http';

import { ConfigService } from './config.service';

import { ChatUserResponse, FriendsRelations, AcceptFriendRequest, FriendsRequest, ConfirmFriendRequest, AddFriendsRequest, MessageResponse, MessagesRequest, GetUserMessagesRequest } from '../models';

import { CurrentUserResponse } from '../models/currentuser.response';

@Injectable()

export class ApiService {

constructor(private http: HttpClient, private config: ConfigService) {

}

getFriendsRequests(): Observable<FriendsRequest[]> {

return this.http.get<FriendsRequest[]>(`${this.config.app.api.url}Chat/GetFriendsRequests`);

}

getNotConfirmedRequests(): Observable<FriendsRequest[]> {

return this.http.get<FriendsRequest[]>(`${this.config.app.api.url}Chat/GetNotConfirmedRequests`);

}

getFriends(): Observable<FriendsRelations> {

return this.http.get<FriendsRelations>(`${this.config.app.api.url}Chat/GetFriends`);

}

getAllUsers(): Observable<ChatUserResponse[]> {

return this.http.get<ChatUserResponse[]> (`${this.config.app.api.url}Chat/GetAllUsers`);

}

getCurrentUser(): Observable<CurrentUserResponse> {

return this.http.get<CurrentUserResponse> (`${this.config.app.api.url}Chat/GetCurrentUser`);

}

acceptFriendKey(acceptFriendRequest: AcceptFriendRequest): Observable<boolean> {

return this.http.post<boolean>(`${this.config.app.api.url}Chat/AcceptFriendRequest`, acceptFriendRequest);

}

confirmFriendRequest(confirmFriendRequest: ConfirmFriendRequest): Observable<FriendsRequest> {

return this.http.post<FriendsRequest>(`${this.config.app.api.url}Chat/ConfirmFriendRequest`, confirmFriendRequest);

}

addFriendsRequests(addFriendsRequest: AddFriendsRequest): Observable<FriendsRequest> {

return this.http.post<FriendsRequest>(`${this.config.app.api.url}Chat/AddFriendsRequests`, addFriendsRequest);

}

synchronizationRequests(): Observable<FriendsRequest[]> {

return this.http.get<FriendsRequest[]>(`${this.config.app.api.url}Chat/SynchronizationRequests`);

}

deleteNotSynchroRequests(requestsToDelete: FriendsRequest[]): Observable<boolean> {

return this.http.post<boolean>(`${this.config.app.api.url}Chat/DeleteNotSynchroRequests`, requestsToDelete);

}

getUserMessages(getUserMessagesRequest: GetUserMessagesRequest): Observable<MessageResponse[]> {

return this.http.post<MessageResponse[]>(`${this.config.app.api.url}Chat/GetUserMessages`, getUserMessagesRequest);

}

sendUserMessage(message: MessagesRequest): Observable<number> {

return this.http.post<number>(`${this.config.app.api.url}Chat/SendUserMessage`, message);

}

}

ПРИЛОЖЕНИЕ Д

Сервис common.service.ts

import { Injectable } from '@angular/core';

import { Observable, Subject } from 'rxjs';

import { ChatUserResponse, FriendsRequest, ConfirmFriendRequest, AcceptFriendRequest } from '../models';

import { ApiService } from './api.service';

import { CurrentUserResponse } from '../models/currentuser.response';

import { THIS\_EXPR } from '@angular/compiler/src/output/output\_ast';

import { CryptoService } from './crypto.service';

@Injectable()

export class CommonService {

public onSelectedFriend = new Subject<ChatUserResponse>();

public onSelectedFriend$ = this.onSelectedFriend.asObservable();

public currentUser: CurrentUserResponse;

public enteredText: string;

constructor(private api: ApiService, private crypto: CryptoService) {

}

public synchronizationLocal(requests: FriendsRequest[]) {

const localItems = Object.keys(localStorage);

localItems.forEach((item) => {

const localItem = localStorage.getItem(item);

const splitIds = localItem.split('|');

if (splitIds && splitIds.length == 3 && !requests.find((req) => (req.requestUserId === splitIds[1] && req.friendId === splitIds[2])

|| (req.requestUserId === splitIds[2] && req.friendId === splitIds[1]))) {

localStorage.removeItem(item);

}

});

}

public async parseFriendsRequests(req: FriendsRequest) {

const requestPublicKey = JSON.parse(req.requestPublicKey) as JsonWebKey;

const rsaImported = await this.crypto.importRSAPublicKey(requestPublicKey);

const aesKey = await this.crypto.generateAESKey();

const aesExported = await this.crypto.exportAESKey(aesKey);

const aesKeyAB = this.crypto.stringToArray(JSON.stringify(aesExported))

const encryptedAesBuffer = new Uint8Array(await this.crypto.encryptRSA(aesKeyAB, rsaImported));

const base64Aes = this.crypto.arrayBufferTobase64(encryptedAesBuffer);

const confirmFriendRequest: ConfirmFriendRequest = { friendId: req.requestUserId, encryptedKey: base64Aes }

this.api.confirmFriendRequest(confirmFriendRequest).subscribe((resp) => {

localStorage.setItem(`Request|${this.currentUser.userId}|${req.requestUserId}`, JSON.stringify([requestPublicKey]));

localStorage.setItem(`Encrypt|${this.currentUser.userId}|${req.requestUserId}`, JSON.stringify(aesExported));

});

}

public async parseNotConfirmedKeys(requests: FriendsRequest[]) {

for (let req of requests) {

const currnetRsaKeys = localStorage.getItem(`Request|${req.friendId}|${req.requestUserId}`);

const currentRsaPublicKeyJSON = JSON.parse(currnetRsaKeys)[0] as JsonWebKey;

const currentRsaPrivateKeyJSON = JSON.parse(currnetRsaKeys)[1] as JsonWebKey;

const currentRsaPrivateKey = await this.crypto.importRSAPrivateKey(currentRsaPrivateKeyJSON);

if (currentRsaPrivateKey) {

let encryptedAesBuffer = this.crypto.base64ToArrayBuffer(req.encryptionKey);

const decryptedAesBuffer = new Uint8Array(await this.crypto.decryptRSA(encryptedAesBuffer, currentRsaPrivateKey));

const decryptedAesString = this.crypto.arrayToString(decryptedAesBuffer);

const aesJsonKey = JSON.parse(decryptedAesString) as JsonWebKey;

const acceptFriendKey: AcceptFriendRequest = { friendId: req.requestUserId };

this.api.acceptFriendKey(acceptFriendKey).subscribe((resp) => {

if (resp) {

localStorage.setItem(`Encrypt|${this.currentUser.userId}|${req.requestUserId}`, JSON.stringify(aesJsonKey));

}

});

}

}

}

}

ПРИЛОЖЕНИЕ Е

Компонента chat.component.ts

import { Component, OnInit, OnDestroy, AfterViewInit, SecurityContext } from '@angular/core';

import { AuthorizeService } from 'src/api-authorization/authorize.service';

import { CryptoService } from '../services/crypto.service';

import { ApiService } from '../services/api.service';

import { Subscription } from 'rxjs';

import { take } from 'rxjs/operators';

import { Message, ChatUserResponse, AcceptFriendRequest, MessagesRequest, MessageFile, GetUserMessagesRequest } from '../models';

import { CommonService } from '../services/common.service';

import { HubConnection, HubConnectionBuilder } from '@aspnet/signalr';

import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';

@Component({

selector: 'chat',

templateUrl: './chat.component.html',

styleUrls: ['./chat.component.scss']

})

export class ChatComponent implements OnDestroy, AfterViewInit {

public IsAuth: boolean;

public authSubscription: Subscription;

public messages: Message[] = [];

private hubConnection: HubConnection;

public selectedFriend: ChatUserResponse;

public currentEmoji: string;

public chatHeader: Element;

private encryptionKey: JsonWebKey;

private publicKey: JsonWebKey;

public isLoadingMessages: boolean;

constructor(private auth: AuthorizeService, private crypto: CryptoService, private api: ApiService, public common: CommonService,private sanitizer: DomSanitizer ) {

this.authSubscription = auth.isAuthenticated().pipe(take(1)).subscribe((isAuth) => {

this.IsAuth = isAuth;

if (this.IsAuth) {

this.auth.getAccessToken().subscribe((token) => {

this.hubConnection = new HubConnectionBuilder()

.withUrl("/chatHub", { accessTokenFactory: () => token })

.build();

this.hubConnection.start();

this.hubConnection.on('Receive', async (mes, type) => {

const key = await this.crypto.importAESKey(this.encryptionKey);

const textAB = this.crypto.base64ToArrayBuffer(mes);

const decryptedText = await this.crypto.decryptAES(textAB, key);

let decrytedMessage: Message = null;

if (type == 0) {

decrytedMessage = { id:mes.id, message: this.crypto.arrayToString(decryptedText), isReply: false, date: Date.now.toString(), messageType: type, files: null };

}

else if (type == 1) {

const base64string = this.crypto.arrayToString(decryptedText)

decrytedMessage = {id:mes.id, message: null, isReply: false, date: Date.now.toString(), messageType: type, files: [{ url: this.sanitizer.bypassSecurityTrustResourceUrl(base64string), icon: null, type: base64string.match(/[^:]\w+\/[\w-+\d.]+(?=;|,)/)[0] }] };

}

this.messages.push(decrytedMessage);

});

});

}

});

this.common.onSelectedFriend$.subscribe((selectedFriend) => {

this.selectedFriend = selectedFriend;

this.getFriendInfo();

});

}

onScroll(event: any) {

if (this.messages && this.messages.length && !this.isLoadingMessages && event.target.scrollTop == 0) {

this.getMessagesAfter(this.messages[0]);

}

}

ngAfterViewInit() {

const chatContainer = document.getElementById("chat-container");

if (chatContainer) {

const scrollableContainer = chatContainer.querySelector('.scrollable');

scrollableContainer.addEventListener('scroll', (event) => { this.onScroll(event) });

}

}

getFriendInfo() {

if (this.IsAuth) {

this.messages = [];

const request: GetUserMessagesRequest = { friendId: this.selectedFriend.userId, id: null };

this.encryptionKey = JSON.parse(localStorage.getItem(`Encrypt|${this.common.currentUser.userId}|${this.selectedFriend.userId}`)) as JsonWebKey;

this.publicKey = JSON.parse(localStorage.getItem(`Request|${this.common.currentUser.userId}|${this.selectedFriend.userId}`)) as JsonWebKey;

this.publicKey = this.publicKey ? this.publicKey[0] : null;

if (this.encryptionKey && this.publicKey) {

this.crypto.getEmojiFromString(this.publicKey.n, 10).then((emoji) => {

this.currentEmoji = emoji;

this.setChatHeader();

});

this.api.getUserMessages(request).subscribe(async (messagesResponse) => {

for (let mes of messagesResponse) {

const key = await this.crypto.importAESKey(this.encryptionKey);

const textAB = this.crypto.base64ToArrayBuffer(mes.encryptedMessage);

const decryptedText = await this.crypto.decryptAES(textAB, key);

let decrytedMessage: Message = null;

if (mes.messageType == 0) {

decrytedMessage = {id:mes.id, message: this.crypto.arrayToString(decryptedText), isReply: mes.isReply, date: mes.date, messageType: mes.messageType, files: null };

}

else if (mes.messageType == 1) {

const base64string = this.crypto.arrayToString(decryptedText);

decrytedMessage = { id:mes.id, message: null, isReply: mes.isReply, date: mes.date, messageType: mes.messageType, files: [{ url: this.sanitizer.bypassSecurityTrustResourceUrl(base64string), type: base64string.match(/[^:]\w+\/[\w-+\d.]+(?=;|,)/)[0], icon: null }] };

}

this.messages.unshift(decrytedMessage);

}

});

}

else {

this.setChatHeader(true);

}

}

}

getMessagesAfter(latestMessage: Message) {

try {

this.isLoadingMessages = true;

const request: GetUserMessagesRequest = { friendId: this.selectedFriend.userId, id: latestMessage.id };

this.api.getUserMessages(request).subscribe(async (messagesResponse) => {

for (let mes of messagesResponse) {

const key = await this.crypto.importAESKey(this.encryptionKey);

const textAB = this.crypto.base64ToArrayBuffer(mes.encryptedMessage);

const decryptedText = await this.crypto.decryptAES(textAB, key);

let decrytedMessage: Message = null;

if (mes.messageType == 0) {

decrytedMessage = {id:mes.id, message: this.crypto.arrayToString(decryptedText), isReply: mes.isReply, date: mes.date, messageType: mes.messageType, files: null };

}

else if (mes.messageType == 1) {

const base64string = this.crypto.arrayToString(decryptedText);

decrytedMessage = {id:mes.id, message: null, isReply: mes.isReply, date: mes.date, messageType: mes.messageType, files: [{ url: this.sanitizer.bypassSecurityTrustResourceUrl(base64string), type: base64string.match(/[^:]\w+\/[\w-+\d.]+(?=;|,)/)[0], icon: null }] };

}

this.messages.unshift(decrytedMessage);

}

this.isLoadingMessages = false;

});

} catch {

alert("Failed to load or decrypt messages!");

this.isLoadingMessages = false;

}

}

ngOnDestroy() {

if (this.authSubscription) {

this.authSubscription.unsubscribe();

}

this.common.onSelectedFriend.next(null);

if (this.hubConnection) {

this.hubConnection.stop();

}

}

public async prepareMessage(event: any) {

const message = event.message;

const files = event.files;

if (files && files.length) {

var reader = new FileReader();

reader.readAsDataURL(files[0]);

reader.onload = (e: any) => {

this.sendMessage(e.target.result, 1);

}

}

if (message) {

this.sendMessage(message, 0);

}

}

public async sendMessage(message: string, type: number) {

const key: CryptoKey = await this.crypto.importAESKey(this.encryptionKey);

const messageAB: Uint8Array = this.crypto.stringToArray(message);

const encryptedMessage: Uint8Array = new Uint8Array(await this.crypto.encryptAES(messageAB, key));

const encryptedMessageBase64: string = this.crypto.arrayBufferTobase64(encryptedMessage);

const messageRequest: MessagesRequest = { friendId: this.selectedFriend.userId, message: encryptedMessageBase64, messageType: type };

this.api.sendUserMessage(messageRequest).subscribe((messageId) => {

if (messageId) {

let newMessage: Message = null;

if (type == 0) {

this.hubConnection.invoke("Send", encryptedMessageBase64, this.selectedFriend.userId, type);

newMessage = {id: messageId, message : message, isReply: true, date: Date.now.toString(), messageType: type, files: null };

} else if (type == 1) {

this.hubConnection.invoke("Send", encryptedMessageBase64, this.selectedFriend.userId, type);

newMessage = {id: messageId, message: null, isReply: true, date: Date.now.toString(), messageType: type, files: [{ url: this.sanitizer.bypassSecurityTrustResourceUrl(message), icon: null, type: message.match(/[^:]\w+\/[\w-+\d.]+(?=;|,)/)[0] }] };

}

this.messages.push(newMessage)

}

});

}

public setChatHeader(isNotHaveKeys: boolean = false) {

if (!this.chatHeader) {

const chatContainer = document.getElementById("chat-container");

this.chatHeader = chatContainer.getElementsByClassName("header")[0];

}

if (!isNotHaveKeys) {

this.chatHeader.innerHTML = `<div class="chat-header-container"><div>Email: ${this.selectedFriend.userName}</div><div>Public key: <span class="emoji-container">${this.currentEmoji}</span></div></div>`;

}

else {

this.chatHeader.innerHTML = `<div class="chat-header-container"><div>Not have keys!</div></div>`;

}

}

}

ПРИЛОЖЕНИЕ Ж

Компонента chat.component.html

<div \*ngIf="IsAuth" class="chat">

<div class="friends-container">

<friends></friends>

</div>

<div class="nb-theme-default chat-container">

<nb-chat title="{{selectedFriend ? chatHeader : 'Select user'}}" id="chat-container" size="giant" scrollBottom="false" status="success">

<nb-chat-message \*ngFor="let msg of (messages | search : common.enteredText)" [type]="msg.messageType == 0 ? 'text' : msg.messageType == 1 ? 'file':'text'" [message]="msg.message" [files]="msg.files" [reply]="msg.isReply">

</nb-chat-message>

<nb-chat-form (send)="prepareMessage($event)" [dropFiles]="true">

</nb-chat-form>

</nb-chat>

</div>

</div>

<ng-container \*ngIf="!IsAuth">Please login first!</ng-container>