МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Проектирование информационных систем**

**Тема:** «Объектно-ориентированное моделирование. СТРУКТУРНЫЕ ДИАГРАММЫ UML»

**Цель:** Изучение методологии объектно-ориентированного моделирования средствами UML. Ознакомление с основными принципами объектно-ориентированного проектирования программного обеспечения, получение навыков проектирования структуры информационной системы с применением UML.

Выполнил:

Володькин Н. Д.,

3 курс 7 группа

Проверил:

Якунович А. В.

Минск 2023

**Описание программных средств**

Draw.io — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

Инструмент работает с Google Диск, Google Workspace и Dropbox, глубоко интегрирован и удобен для работы с продуктами Confluence и Jira от Atlassian. Пользователи также могут работать с диаграммами в автономном режиме и сохранять их локально, используя настольное приложение для персональных компьютеров.

Инструмент позволяет создавать: графики, диаграммы, таблицы, презентации, блок-схемы, планы помещений, воронки продаж, ментальные карты, карты сайтов.

**Постановка задачи**

Задачей данной работы выступает описание структуры взаимодействия классов в приложении, описанной в лабораторной работе #1.

Главными сущностями приложения выступают:

* Api – класс с помощью которого осуществляются запросы к API приложения.
* Result<T> – generic класс, который представляет собой ответ на запрос к серверу.
* ChatService – класс, предоставляющий функционал для работы с чатами.
* FilesService – класс, позволяющий загружать вложения на сервер.
* MessagesService – класс, позволяющий работать с сообщениями.
* UsersService – класс для работы с пользователями.

**Описание практического задания**

Разработанная диаграмма классов представлена на рисунке 1.1.

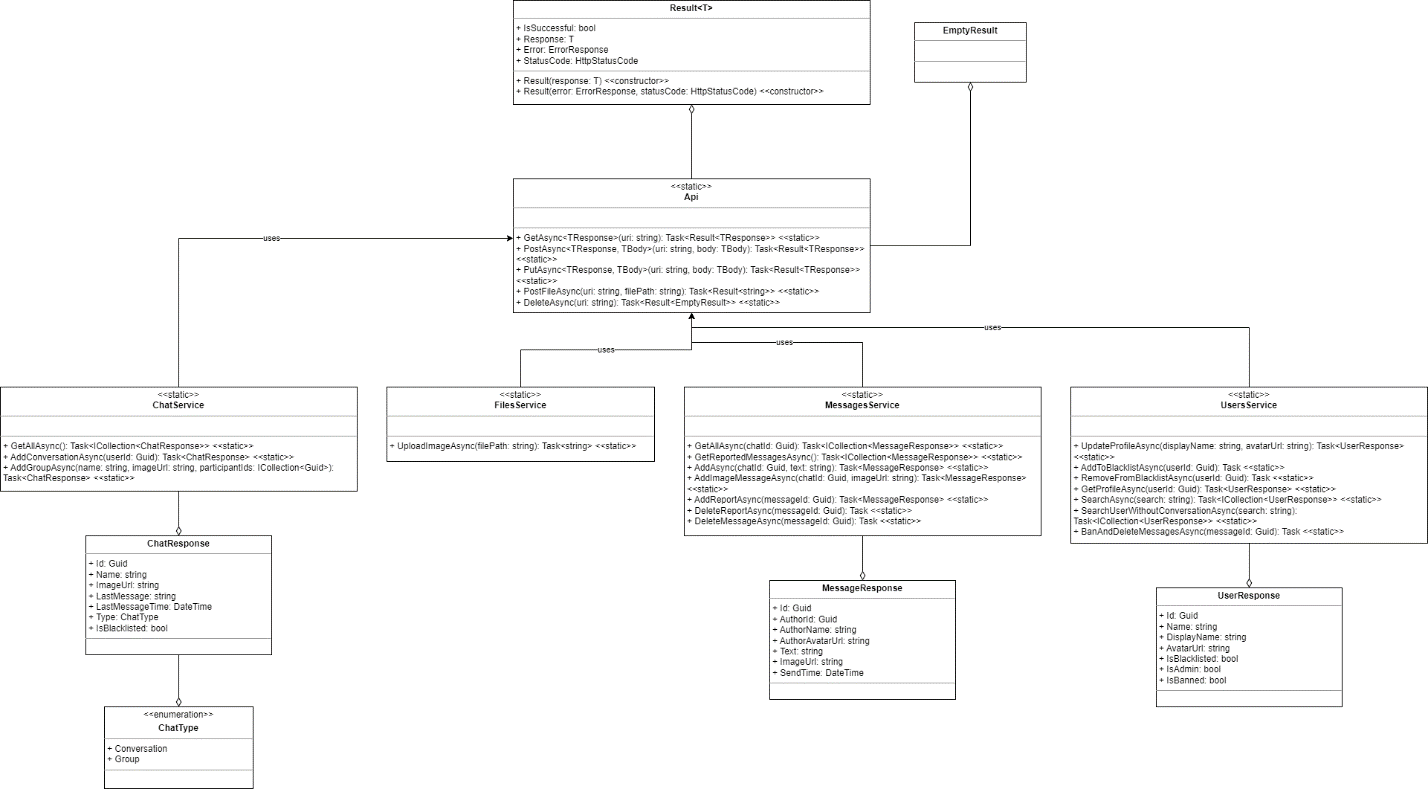


Рисунок 1.1 – Диаграмма классов мессенджера

Api – класс с помощью которого осуществляются запросы к API приложения.

Этот класс имеет следующие методы:

* GetAsync<TResponse>(uri: string) - выполняет асинхронный GET-запрос по указанному URI и возвращает результат в виде задачи, содержащей Result<TResponse>.
* PostAsync<TResponse, TBody>(uri: string, body: TBody) - выполняет асинхронный POST-запрос по указанному URI с телом запроса body и возвращает результат в виде задачи, содержащей Result<TResponse>.
* PutAsync<TResponse, TBody>(uri: string, body: TBody) - выполняет асинхронный PUT-запрос по указанному URI с телом запроса body и возвращает результат в виде задачи, содержащей Result<TResponse>.
* PostFileAsync(uri: string, filePath: string) - выполняет асинхронный POST-запрос файла по указанному URI и возвращает результат в виде задачи, содержащей Result<string>.
* DeleteAsync(uri: string) - выполняет асинхронный DELETE-запрос по указанному URI и возвращает результат в виде задачи, содержащей Result<EmptyResult>.

Далее более подробно будут описаны каждый из модулей диаграммы.

1. ***Модуль «Chat»:***

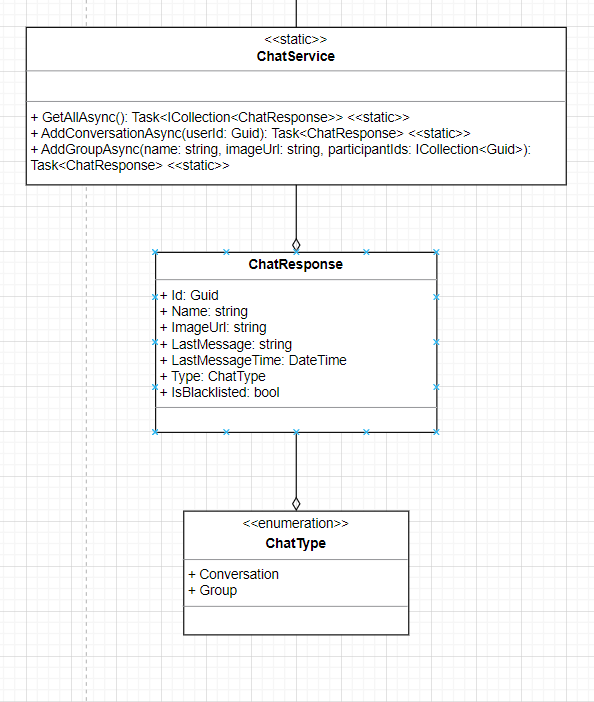


Рисунок 1.2 – Модуль «Chat»

ChatService – класс, предоставляющий функционал для работы с чатами. Он содержит методы:

* GetAllAsync() - выполняет асинхронный запрос на получение всех чатов и возвращает результат в виде задачи, содержащей коллекцию ICollection<ChatResponse>.
* AddConversationAsync(userId: Guid) - выполняет асинхронный запрос на добавление нового разговора с указанным userId и возвращает результат в виде задачи, содержащей ChatResponse.
* AddGroupAsync(name: string, imageUrl: string, participantIds: ICollection<Guid>) - выполняет асинхронный запрос на добавление новой группы с указанными name, imageUrl и participantIds и возвращает результат в виде задачи, содержащей ChatResponse.

ChatResponse – класс, содержащий свойства описывающие пользователя. Далее перечислены все публичные свойства:

* Id: Guid - уникальный идентификатор чата в формате Guid.
* Name: string - имя чата в виде строки.
* ImageUrl: string - URL-адрес изображения чата в виде строки.
* LastMessage: string - последнее сообщение в чате в виде строки.
* LastMessageTime: DateTime - время последнего сообщения в чате в формате DateTime.
* Type: ChatType - тип чата, который может быть одним из значений перечисления ChatType.
* IsBlacklisted: bool - булево значение, указывающее, добавлен ли чат в черный список. Если true, чат добавлен в черный список; если false, чат не добавлен в черный список.
* ChatType – перечисление, которое имеет два значения для группы и личных сообщений.

1. ***Модуль «Files»:***

FilesService – класс, позволяющий загружать вложения на сервер. Он содержит один метод: UploadImageAsync(filePath: string) - выполняет асинхронную загрузку изображения по указанному пути к файлу filePath и возвращает результат в виде задачи, содержащей строку с URL-адресом загруженного изображения.

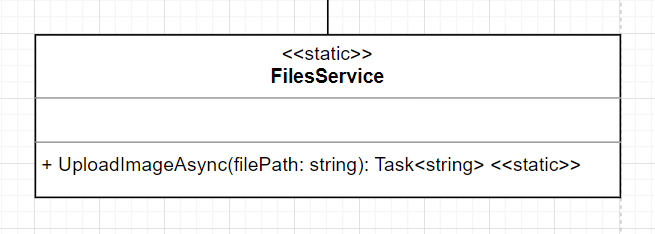


Рисунок 1.3 – Модуль «Files»

1. ***Модуль «Messages»:***

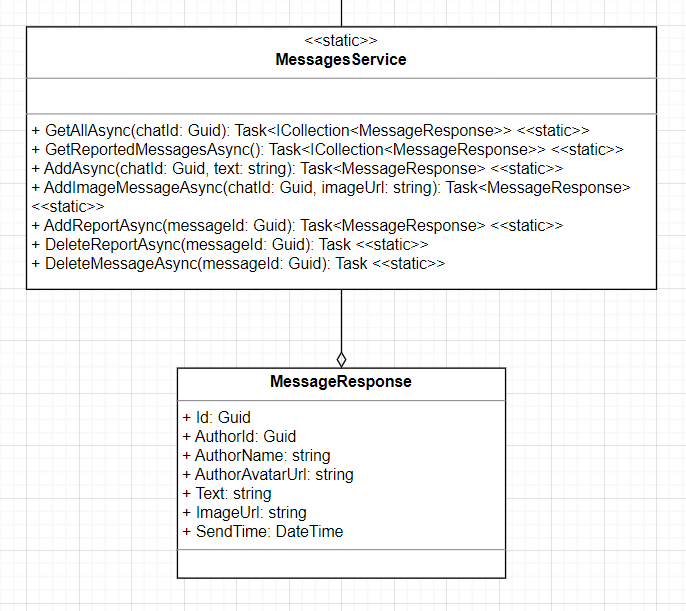


Рисунок 1.4 – Модуль «Messages»

MessagesService – класс, позволяющий работать с сообщениями. Ниже описаны его методы:

* GetAllAsync(chatId: Guid) - выполняет асинхронный запрос на получение всех сообщений в чате с указанным chatId и возвращает результат в виде задачи, содержащей коллекцию ICollection<MessageResponse>.
* GetReportedMessagesAsync() - выполняет асинхронный запрос на получение всех сообщений, которые были отмечены как нежелательные, и возвращает результат в виде задачи, содержащей коллекцию ICollection<MessageResponse>.
* AddAsync(chatId: Guid, text: string) - выполняет асинхронный запрос на добавление нового текстового сообщения text в чат с указанным chatId и возвращает результат в виде задачи, содержащей MessageResponse.
* AddImageMessageAsync(chatId: Guid, imageUrl: string) - выполняет асинхронный запрос на добавление нового сообщения с изображением по указанному URL imageUrl в чат с указанным chatId и возвращает результат в виде задачи, содержащей MessageResponse.
* AddReportAsync(messageId: Guid) - выполняет асинхронный запрос на добавление отчета о сообщении с указанным messageId и возвращает результат в виде задачи, содержащей MessageResponse.
* DeleteReportAsync(messageId: Guid) - выполняет асинхронный запрос на удаление отчета о сообщении с указанным messageId и возвращает результат в виде задачи.
* DeleteMessageAsync(messageId: Guid) - выполняет асинхронный запрос на удаление сообщения с указанным messageId и возвращает результат в виде задачи.

MessageResponse – класс, содержащий свойства описывающие сообщение:

* Id: Guid - уникальный идентификатор сообщения в формате Guid.
* AuthorId: Guid - уникальный идентификатор автора сообщения в формате Guid.
* AuthorName: string - имя автора сообщения в виде строки.
* AuthorAvatarUrl: string - URL-адрес аватара автора сообщения в виде строки.
* Text: string - текст сообщения в виде строки.
* ImageUrl: string - URL-адрес изображения сообщения в виде строки, если таковое имеется.
* SendTime: DateTime - время отправки сообщения в формате DateTime.

1. ***Модуль «Users»:***

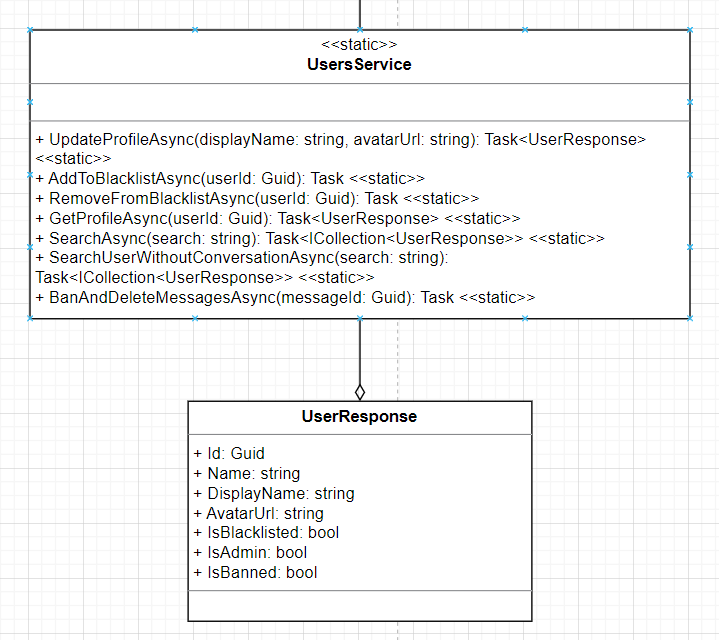


Рисунок 1.5 – Модуль «Users»

UsersService – класс для работы с пользователями. Ниже представлены его методы:

* UpdateProfileAsync(displayName: string, avatarUrl: string) - выполняет асинхронное обновление профиля пользователя с указанными displayName и avatarUrl и возвращает результат в виде задачи, содержащей UserResponse.
* AddToBlacklistAsync(userId: Guid) - выполняет асинхронное добавление пользователя с указанным userId в черный список и возвращает результат в виде задачи.
* RemoveFromBlacklistAsync(userId: Guid) - выполняет асинхронное удаление пользователя с указанным userId из черного списка и возвращает результат в виде задачи.
* GetProfileAsync(userId: Guid) - выполняет асинхронный запрос на получение профиля пользователя с указанным userId и возвращает результат в виде задачи, содержащей UserResponse.
* SearchAsync(search: string) - выполняет асинхронный поиск пользователей по указанной строке search и возвращает результат в виде задачи, содержащей коллекцию ICollection<UserResponse>.
* SearchUserWithoutConversationAsync(search: string) - выполняет асинхронный поиск пользователей, с которыми еще не было разговора, по указанной строке search и возвращает результат в виде задачи, содержащей коллекцию ICollection<UserResponse>.

UserResponse – класс, содержащий свойства описывающие пользователя:

* Id: Guid - уникальный идентификатор пользователя в формате Guid.
* Name: string - имя пользователя в виде строки.
* DisplayName: string - отображаемое имя пользователя в виде строки.
* AvatarUrl: string - URL-адрес аватара пользователя в виде строки.
* IsBlacklisted: bool - булево значение, указывающее, добавлен ли пользователь в черный список. Если true, пользователь добавлен в черный список; если false, пользователь не добавлен в черный список.
* IsAdmin: bool - булево значение, указывающее, является ли пользователь администратором. Если true, пользователь является администратором; если false, пользователь не является администратором.
* IsBanned: bool - булево значение, указывающее, забанен ли пользователь. Если true, пользователь забанен; если false, пользователь не забанен

Далее представлены пакеты, в которые были сгруппированы классы.

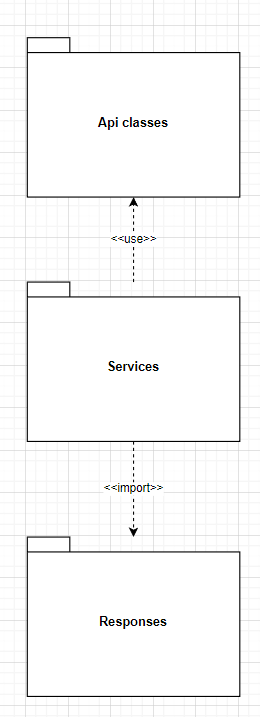


Рисунок 1.6 – Пакеты классов

Responses содержит классы-модели, которые содержат информацию о различных обьектах.

Services содержит основную логику для взаимодействия с API приложения.

Api classes содержат классы-помошники для упрощения работы с API.

**Вывод:** Я изучил методологию объектно-ориентированного моделирования средствами UML. Ознакомилсь с основными принципами объектно-ориентированного проектирования программного обеспечения, получил навыки проектирования функциональности информационной системы с применением UML.