**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»**

**Институт приборостроения, автоматизации и информационных технологий**

**Кафедра информационных систем и цифровых технологий**

**ОТЧЕТ**

по производственной практике

На материалах ООО «Стерх-2»

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шорин В.Д.

Группа 71ПГ

Направление 090304 Программная инженерия

Руководитель практики   
от университета \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фролов А.И.

Руководитель практики   
от профильной организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федин В.Н.

Оценка защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Орел 2021**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»**

**Институт приборостроения, автоматизации и информационных технологий**

**Кафедра информационных систем и цифровых технологий**

Направление подготовки 090304 Программная инженерия

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на производственную практику**

для обучающегося Шорина Владислава Дмитриевича

4 курса очной формы обучения, группы 71ПГ

Место прохождения практики: ООО «Стерх-2»

Срок прохождения практики с «11» мая 2021 г. по «7» июня 2021 г.

**Содержание практики (вопросы, подлежащие изучению):**

* Анализ задачи разработки специализированного сервиса обмена сообщениями в ЕОИС ОГУ им. И.С. Тургенева;
* Определение спецификаций к сервису обмена сообщениями;
* Проектирование серверной подсистемы сервиса обмена сообщениями;
* Реализация, тестирование и отладка программного обеспечения.

**Планируемые результаты практики:**

* Спецификации к программному обеспечению;
* Спроектированные структура, база данных, API и алгоритмы;
* Реализованная сервисная подсистема.

Руководитель практики от университета \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Фролов

Согласовано:

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Федин

Задание принял: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 202\_\_ г.

**СОВМЕСТНЫЙ РАБОЧИЙ ГРАФИК (ПЛАН)**

**проведения производственной практики**

Обучающегося 4 курса очнойформы обучения, группы 71ПГ

Направление подготовки / специальность 090304 Программная инженерия

Место прохождения практики ООО «Стерх-2»

Срок прохождения практики с «11» мая 2021 г. по «7» июня 2021 г.

| № п/п | Наименование этапа проведения практики\* | Вид работ | Срок прохождения этапа практики | Форма отчетности | Отметка о выполнении |
| --- | --- | --- | --- | --- | --- |
| 1 | Подготовительный (организационный) этап | 1.Организационное собрание для разъяснения целей, задач, содержания и порядка прохождения практики.  2. Инструктаж по технике безопасности.  3. Выдача индивидуального задания. | 11.05.2021 | Индивидуальное задания |  |
| 2 | Основной этап | 1. Ознакомление с конкретными видами деятельности в соответствии с положениями структурных подразделений и должностными инструкциями.  2. Сбор информации и материалов практики.  3. Выполнение программы практики, индивидуального задания на практику.  4.Обработка, систематизация и анализ фактического и теоретического материала. | 12.05.2021  -  5.06.2021 | Фактический и теоретический материал, результаты анализа |  |
| 3 | Заключительный этап | Составление отчета по практике.  Защита отчета по практике с представлением материалов конкретной профильной организации. | 7.06.2021 | Отчет по практике |  |

\* Этапы проведения практики и виды работ по ним могут меняться в зависимости от направления подготовки

Руководитель практики от университета \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Фролов

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Федин

С рабочим графиком (планом) ознакомлен:

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Д. Шорин

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc73650920)

[1 АНАЛИЗ ЗАДАЧИ РАЗРАБОТКИ СПЕЦИАЛИЗИРОВАННОГО СЕРВИСА ОБМЕНА СООБЩЕНИЯМИ В ЕОИС ОГУ им. И.С. Тургенева 7](#_Toc73650921)

[1.1 Структура сервисов личных кабинетов студента и преподавателя ЕОИС ОГУ им. И.С. Тургенева 7](#_Toc73650922)

[1.2 Функциональные возможности существующей системы обмена сообщениями 12](#_Toc73650923)

[1.3 Анализ аналогичных решений и возможностей применения существующих программных средств 13](#_Toc73650924)

[1.4 Постановка задачи и формулировка требований 23](#_Toc73650925)

[1.4.1 Функциональные возможности сервиса обмена сообщениями 24](#_Toc73650926)

[1.4.2 Нефункциональные требования к серверной подсистеме 24](#_Toc73650927)

[2 ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ К СЕРВИСУ ОБМЕНА СООБЩЕНИЯМИ 25](#_Toc73650928)

[2.1 Разработка архитектуры 25](#_Toc73650929)

[2.2 Разработка функциональных спецификаций 26](#_Toc73650930)

[2.3 Разработка информационных спецификаций серверной подсистемы сервиса обмена сообщениями 34](#_Toc73650931)

[2.4 Логическая модель API 35](#_Toc73650932)

[3 ПРОЕКТИРОВАНИЕ СЕРВЕРНОЙ ПОДСИСТЕМЫ СЕРВИСА ОБМЕНА СООБЩЕНИЯМИ 41](#_Toc73650933)

[3.1 Проектирование структуры программного обеспечения 41](#_Toc73650934)

[3.2 Проектирование основных компонентов программного обеспечения 41](#_Toc73650935)

[3.3 Проектирование API 44](#_Toc73650936)

[3.4 Проектирование базы данных 46](#_Toc73650937)

[3.5 Проектирование алгоритмов 49](#_Toc73650938)

[4 РЕАЛИЗАЦИЯ, ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 52](#_Toc73650939)

[4.1 Реализация основных компонентов серверной подсистемы 52](#_Toc73650940)

[4.1.1 Класс-контроллер ChatController 52](#_Toc73650941)

[4.1.2 Класс-модель Chat 54](#_Toc73650942)

[4.2 Реализация API 55](#_Toc73650943)

[4.3 Тестирование и отладка программного обеспечения 56](#_Toc73650944)

[ЗАКЛЮЧЕНИЕ 61](#_Toc73650945)

[СПИСОК ЛИТЕРАТУРЫ 62](#_Toc73650946)

[Приложение 1](#_Toc73650947) [Документация API 63](#_Toc73650948)

[Приложение 2](#_Toc73650949) [Исходный текст программы 76](#_Toc73650950)

# ВВЕДЕНИЕ

Важной частью жизни человека является общение. В современном мире невозможно представить жизнь человека без использования какого-либо гаджета: телефона, планшета, компьютера и т.д. Поэтому, большая часть общения между людьми перешла именно в наши технические средства и сервисы обмена сообщениями, или мессенджеры, стали неотъемлемой их частью. Данные сервисы позволяют человеку поддерживать контакт с важными в его жизни людьми в практически любой момент и на любом расстоянии и обмениваться с ними почти любой информацией.

Сервисы обмена сообщениями играют важную роль не только в потребительской сфере, но также и в деловой. Это позволяет упростить рабочий процесс посредством ускорения обмена необходимой информацией между соответствующими группами людей.

Также, сервисы обмена сообщениями являются основными средствами коммуникации в процессе образования, особенно дистанционного. Они позволяют упростить процессы общения между студентом и преподавателем, выполнения различных контрольных работ, лабораторных работ и т.п.

Целью практики является сокращение времени, затрачиваемого пользователями на выполнение основных операций при использовании сервиса обмена сообщениями ЕОИС ОГУ им. И.С. Тургенева.

Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать задачу разработки специализированного сервиса обмена сообщениями;
2. изучить функциональные возможности существующего решения;
3. сформулировать требования и определить функциональные и нефункциональные возможности разрабатываемой серверной подсистемы;
4. разработать архитектуру серверной подсистемы и базы данных сервиса обмена сообщениями;
5. разработать функциональные и информационные спецификации;
6. спроектировать собственное API сервера и основные алгоритмы;
7. реализовать программный продукт.

# АНАЛИЗ ЗАДАЧИ РАЗРАБОТКИ СПЕЦИАЛИЗИРОВАННОГО СЕРВИСА ОБМЕНА СООБЩЕНИЯМИ В ЕОИС ОГУ им. И.С. Тургенева

## Структура сервисов личных кабинетов студента и преподавателя ЕОИС ОГУ им. И.С. Тургенева

Вслед за развитием информационных технологий во всем мире и расширений перечня возможностей использования различных технических средств претерпевает существенные изменения и сфера образования. Немаловажную роль играют и непредвиденные обстоятельства, которые вынуждают образовательные учреждения использовать дистанционные технологии. Вследствие совокупности данных факторов вытекает необходимость внедрения современных и удобных средств реализации всех образовательных процессов и технологий. К таким средствам можно отнести: официальные сайты образовательных учреждений со всей необходимой информацией, личные кабинеты студентов и преподавателей, средства обмена информацией между участниками образовательного процесса.

Для создания современных и практичных условий проведения образовательного процесса в ОГУ им. И.С. Тургенева создан соответствующий официальный сайт. На данном сайте для участников образовательного процесса создан личный кабинет, для студента и преподавателя соответственно, необходимый для того, чтобы:

* получать актуальную информацию об учебном процессе в короткий промежуток времени;
* осуществлять взаимодействие студента с профессорско-преподавательским составом;
* иметь доступ к информации в электронных библиотечных системах и электронных образовательных ресурсах, связанный с высшим учебным заведением и многие другие функции.

Для, того, чтобы воспользоваться всем доступным функционалом личного кабинета, в первую очередь, необходимо являться участником образовательного процесса в ОГУ им. И.С. Тургенева (т.е. быть студентом или членом профессорско-преподавательского состава). При соблюдении данного условия, получение доступа личному кабинету студента или преподавателя сводится к авторизации в личном кабинете, т.е. ввод персональных данных участника в соответствующей графе: для студента это номер зачетной книжки, фамилия, имя и пароль. Для преподавателя данный процесс аналогичен.

При входе в учетную запись, пользователю предоставляются поля с его личной информацией, а также список доступных функциональных возможностей. Структура сервисов личных кабинетов студента и преподавателя будет иметь некоторые различия.

На рис. 1.1 представлен интерфейс веб-версии личного кабинета студента ОГУ им. И.С. Тургенева.

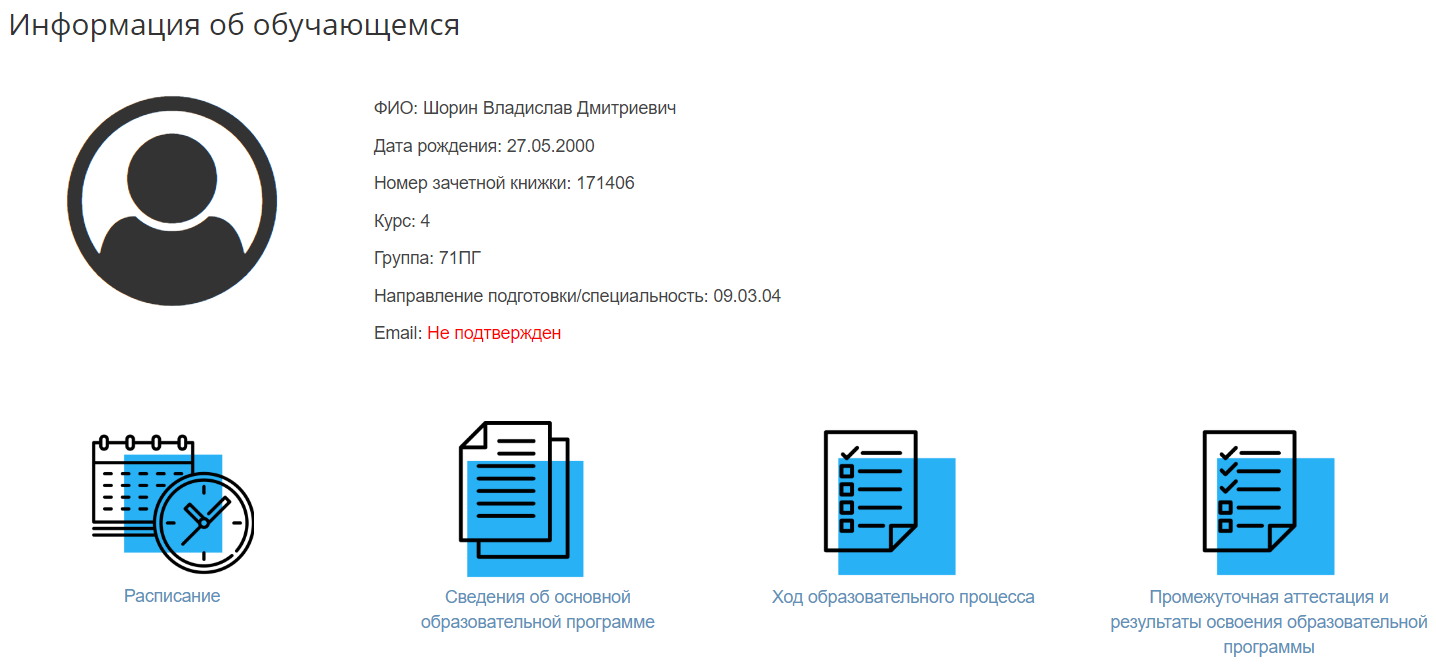


Рисунок 1.1 – Интерфейс веб-версии личного кабинета студента

Вначале представлена личная информация, идентифицирующая студента (фотография (при наличии), ФИО, курс обучения и т.д.). Далее идет список доступных ему функциональных возможностей:

* расписание – обеспечивает доступ к актуальному расписанию занятий, отображая для каждого ФИО преподавателя, название предмета, тип занятия, время проведения, место проведения;

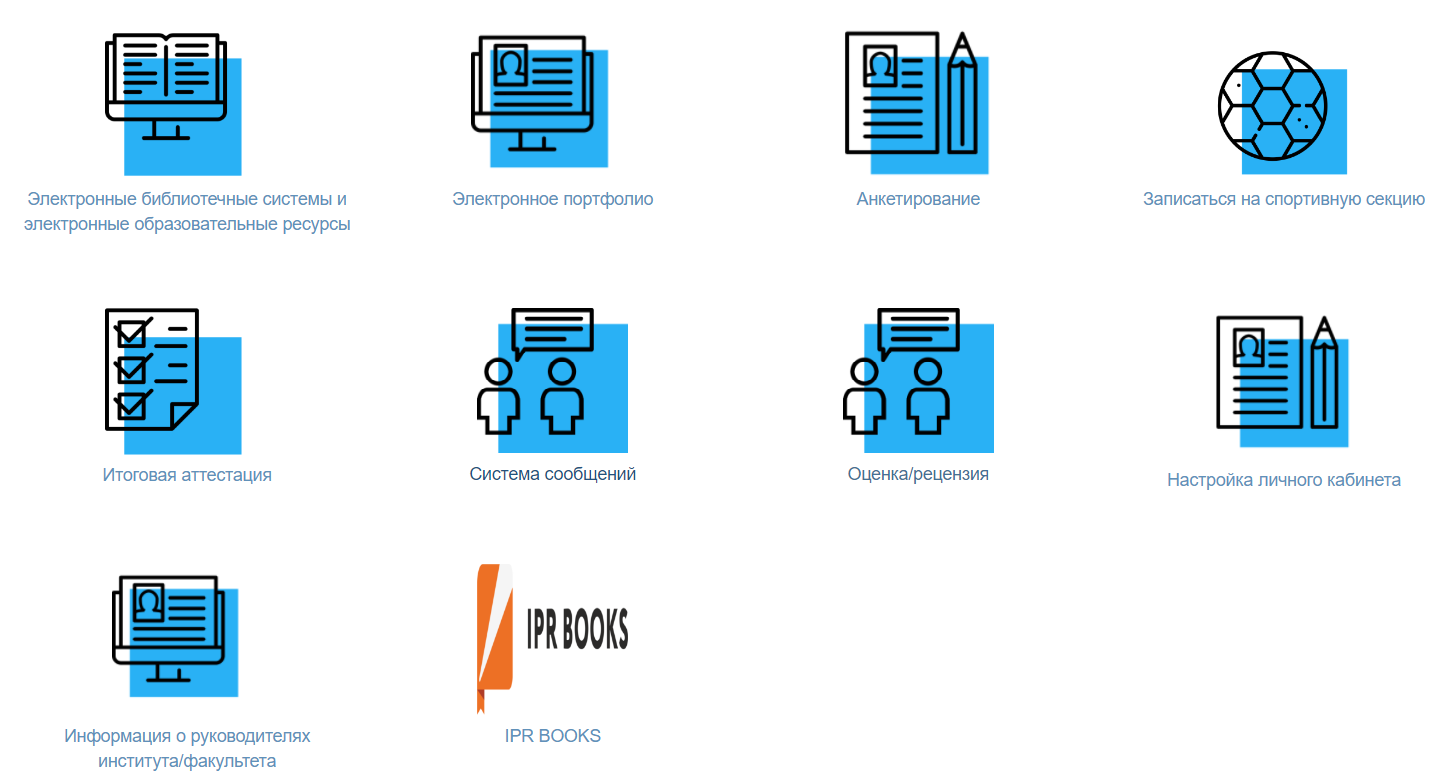


Рисунок 1.1 – Интерфейс веб-версии личного кабинета студента (продолжение)

* сведения об основной образовательной программе – включает в себя сведения об образовательной программе, рабочие программы по предметам, программы практик, ГИА, методические материалы;
* ход образовательного процесса – показывает текущие результаты освоения рабочей программы, выраженные в баллах балльно-рейтинговой системы;
* промежуточная аттестация и результаты освоения образовательной программы – представляет студенту результаты промежуточной аттестации за каждый прошедший семестр и текущей сессии;
* электронные библиотечные системы и электронные образовательные ресурсы – предоставляет доступ к электронной версии научной библиотеки ОГУ им. И.С. Тургенева;
* электронное портфолио – содержит портфолио студента как в учебной и научно-исследовательской, так и во внеучебной деятельности (дипломы, грамоты, курсовые работы и т.д.);
* анкетирование – позволяет студенту пройти анкетирование для оценки качества учебного процесса в высшем учебном заведении;
* записаться на спортивную секцию – дает доступ студенту к возможности записи на интересующую его спортивную секцию (только для тех, у которых в учебном плане на следующий семестр предусмотрены занятия по физической культуре);
* итоговая аттестация – содержит результаты государственной итоговой аттестации;
* система сообщений – сервис обмена сообщениями между студентами и преподавателями внутри университета;
* оценка/рецензия – позволяет оставить отзыв или рецензию на ход образовательного процесса;
* настройка личного кабинета – предоставляет возможность смены электронной почты и пароля личного кабинета;
* информация о руководителях института/факультета – предоставляет информацию о руководителях института или факультета (ФИО, должность, номер телефона, почта);
* IPR BOOKS – позволяет перейти на портал IPR BOOKS.

На рисунке 1.2 представлен интерфейс веб-версии личного кабинета преподавателя ОГУ им. И.С. Тургенева. Вначале представлена личная информация о преподавателе (ФИО, фотография (при наличии), занимаемые должности, ученая степень, ученое звание, контактная информация). Во вкладке «Информация о преподавателе» содержатся сведения об образовании, квалификации, повышениях квалификации и профессиональной переподготовки, общем стаже работы и стаже работы по специальности, а также список преподаваемых дисциплин. Возможность редактирования информации представляется путем нажатия кнопки «Редактировать».

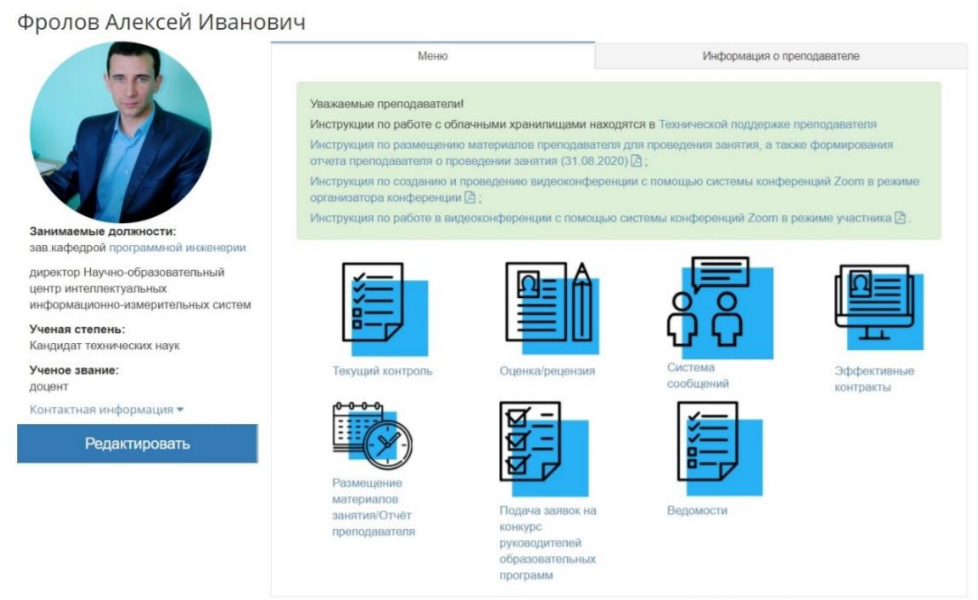


Рисунок 1.2 – Интерфейс веб-версии личного кабинета преподавателя

Вкладка «Меню» содержит список доступных функциональных возможностей:

* текущий контроль – позволяет выставлять текущие результаты освоения рабочей программы, выраженные в баллах балльно-рейтинговой системы, студентам учебных групп;
* оценка/рецензия - позволяет оставить отзыв или рецензию о ходе образовательного процесса;
* система сообщений – сервис обмена сообщениями между студентами и преподавателями внутри университета;
* эффективные контракты – позволяет размещать сведения об эффективных контрактах;
* размещение материалов занятий / отчет преподавателя – позволяет прикреплять методические материалы к занятию и предоставлять отчет о проведенном занятии;
* подача заявок на конкурс руководителей образовательных программ – сервис подачи заявок;
* ведомости – позволяет заполнять и предоставлять зачетные и экзаменационные ведомости в деканат.

## Функциональные возможности существующей системы обмена сообщениями

У студента и преподавателя отличаются не только личные кабинеты в целом, но и некоторые их части, в том числе система обмена сообщениями. У каждой стороны образовательного процесса имеются как общие функциональные возможности, так и индивидуальные (они в большей степени присущи профессорско-преподавательскому составу).

Студент в своем личном кабинете выбирает пункт «Система сообщений», после чего попадает на страницу сервиса обмена сообщениями (рис. 1.3).

В существующей системе обмена сообщениями данному участнику образовательного процесса доступны следующие функциональные возможности:

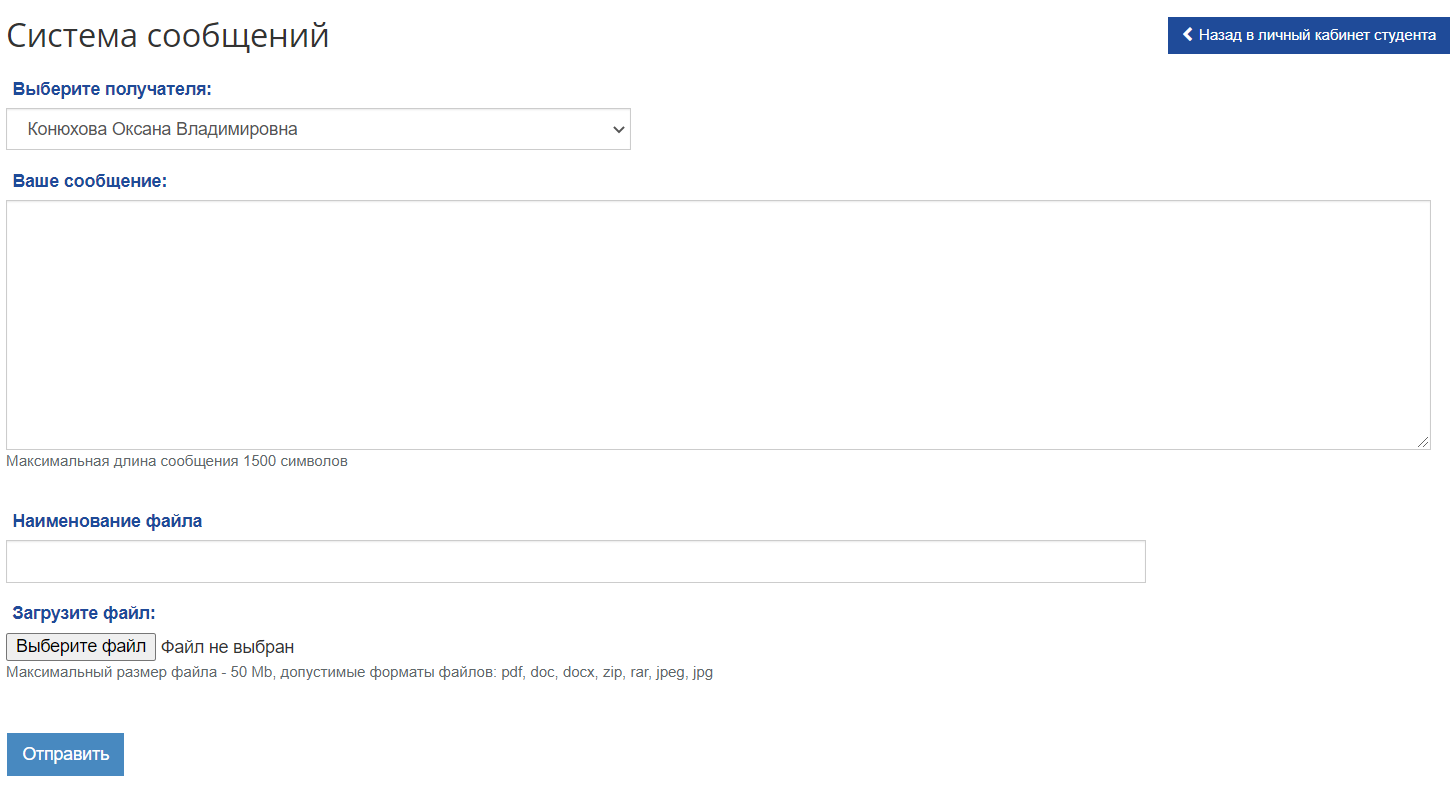


Рисунок 1.3 – Интерфейс веб-версии системы сообщений студента

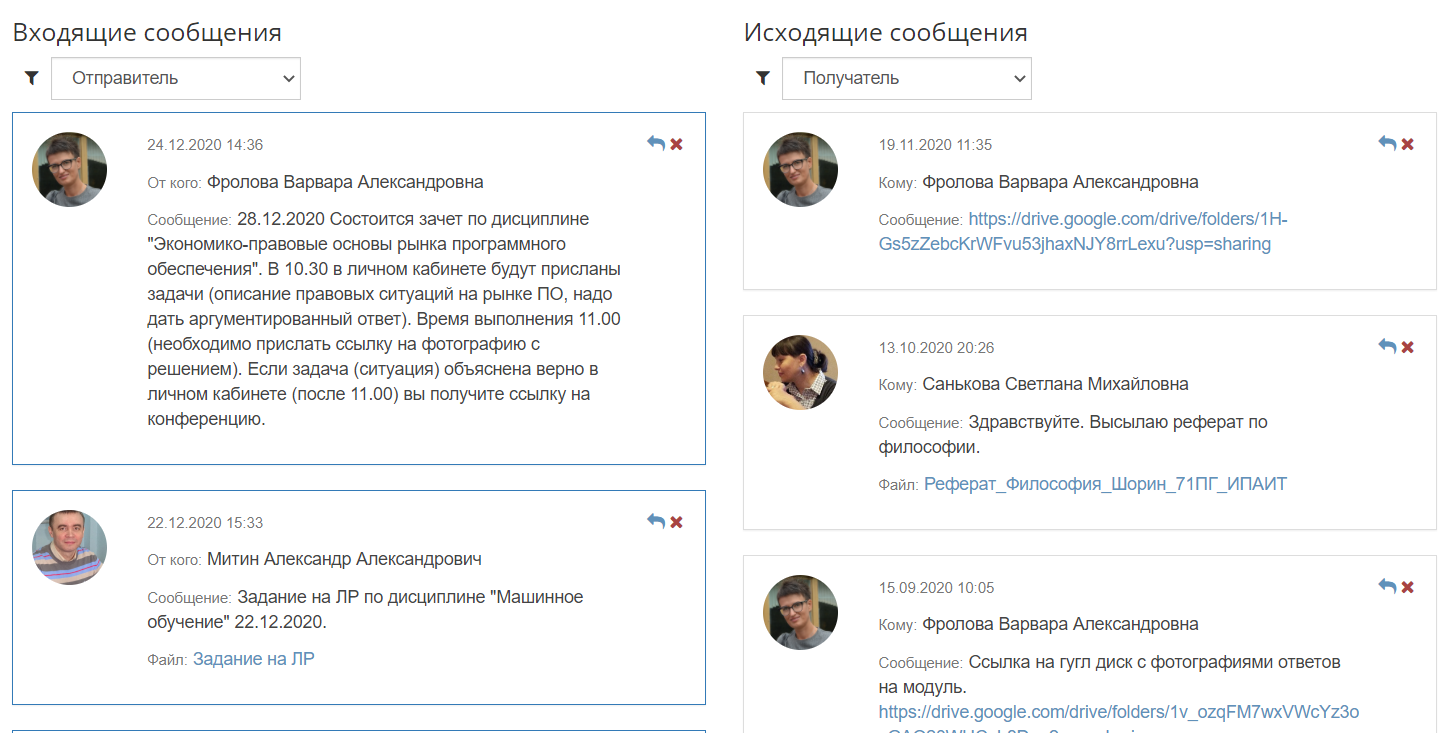


Рисунок 1.3 – Интерфейс веб-версии системы сообщений студента (продолжение)

* выбор получателя сообщения из списка преподавателей, которые закреплены за дисциплинами в текущем семестре;
* набор текстового сообщения получателю;
* прикрепление файла с дополнительными материалами в одном из доступных форматов (pdf, dox, docx, zip, rar, jpeg, jpg);
* просмотр списка входящих сообщений;
* сортировка входящих сообщений (от всех отправителей; от конкретного преподавателя)
* просмотр списка отправленных сообщений;
* сортировка отправленных сообщений (для всех получателей; для конкретного преподавателя)

## Анализ аналогичных решений и возможностей применения существующих программных средств

В целях изучения актуальности разрабатываемой программной системы, перед началом разработки требуется провести поиск и сравнение уже существующих решений по данной тематике, а также выделить необходимые критерии сравнения.

Исходя из того факта, что программное обеспечение разрабатывается непосредственно для нужд ОГУ им. И.С. Тургенева, конкурентоспособных аналогов, соответствующих выявленным функциональным и нефункциональным требованиям в полной мере, не существует. Однако, мы можем рассмотреть программное обеспечение, чей функционал наиболее близок к требуемому.

В открытых источниках не имеется информации о сервисах обмена сообщениями, функционал которых в большей степени предназначен для удовлетворения потребностей высших учебных заведений, а не широкого круга лиц, потому что данные сервисы входят в состав функциональных возможностей личных кабинетов соответствующих высших учебных заведений, доступ к которым достаточно проблематично получить, не являясь участником образовательного процесса соответствующего университета. Поэтому для изучения и сравнения с разрабатываемой программной системой, помимо существующего веб-решения, будут выбраны такие популярные мессенджеры, как «WhatsApp» и «Viber».

Данный анализ позволит, в первую очередь, выявить наиболее востребованные направления в функциональной части современных сервисов обмена сообщениями. Результаты проведенного изучения аналогов можно использовать для последующей возможной корректировки функциональных и нефункциональных требований, выявленных ранее. Выделим необходимые критерии, по которым будет производиться сравнительный анализ в рамках серверной подсистемы сервиса обмена сообщениями:

1. возможность создания индивидуального чата;
2. возможность создания группового чата;
3. возможность объединения чатов в совокупности по каким-либо критериям;
4. возможность обмена материалами (аудио- и видеоформата, изображения, электронными документы и т.д.);
5. простота интеграции в систему управления учебным процессом высшего учебного заведения (в данном случае, ОГУ им. И.С. Тургенева);
6. стек технологий, применяемых для реализации серверной подсистемы (применяемые языки программирования, фреймворки, СУБД и т.д.);
7. тип архитектуры сервера.

В первую очередь, необходимо рассмотреть существующее web-решение ОГУ им. И.С. Тургенева. Данная система сообщений позволяет студенту:

* выбрать получателя сообщения из списка доступных преподавателей (список формируется на основе загруженной в систему нагрузки на преподавателей и студентов. Также может дополняться при необходимости по усмотрению администраторов);
* ввести текст сообщения и прикрепить к нему документ с названием (доступные форматы документов: pdf, doc, docx, zip, rar, jpeg, jpg);
* отсортировать списки отправленных и полученных сообщений по преподавателю.

Для преподавателя отличие в функциональных возможностях от студента отличается только в наличии возможности выбора адресата сообщения между целой группой или конкретным студентом данной группы. Соответственно, и фильтрация отправленных и полученных сообщений производится по данным критериям.

Технические характеристики данного решения:

* язык программирования – PHP;
* фреймворк – Zend;
* база данных – MS SQL Server;
* операционная система – CentOS;
* веб-сервер – Apache;
* протокол – HTTP.

На рис. 1.4 представлен интерфейс веб-версии системы сообщений ОГУ им. И.С. Тургенева.

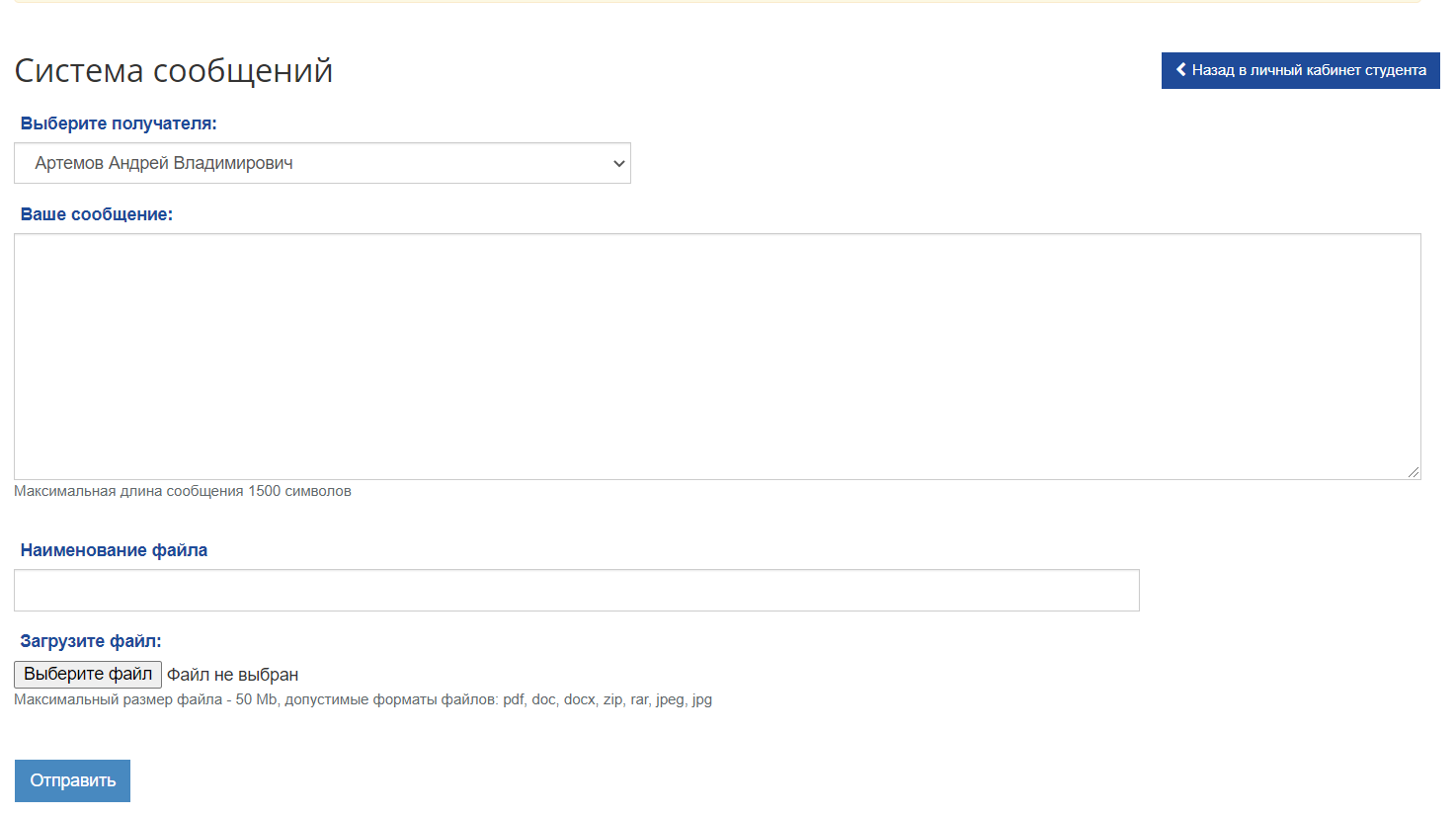


Рисунок 1.4 – Интерфейс веб-версии системы сообщений ОГУ им. И.С. Тургенева

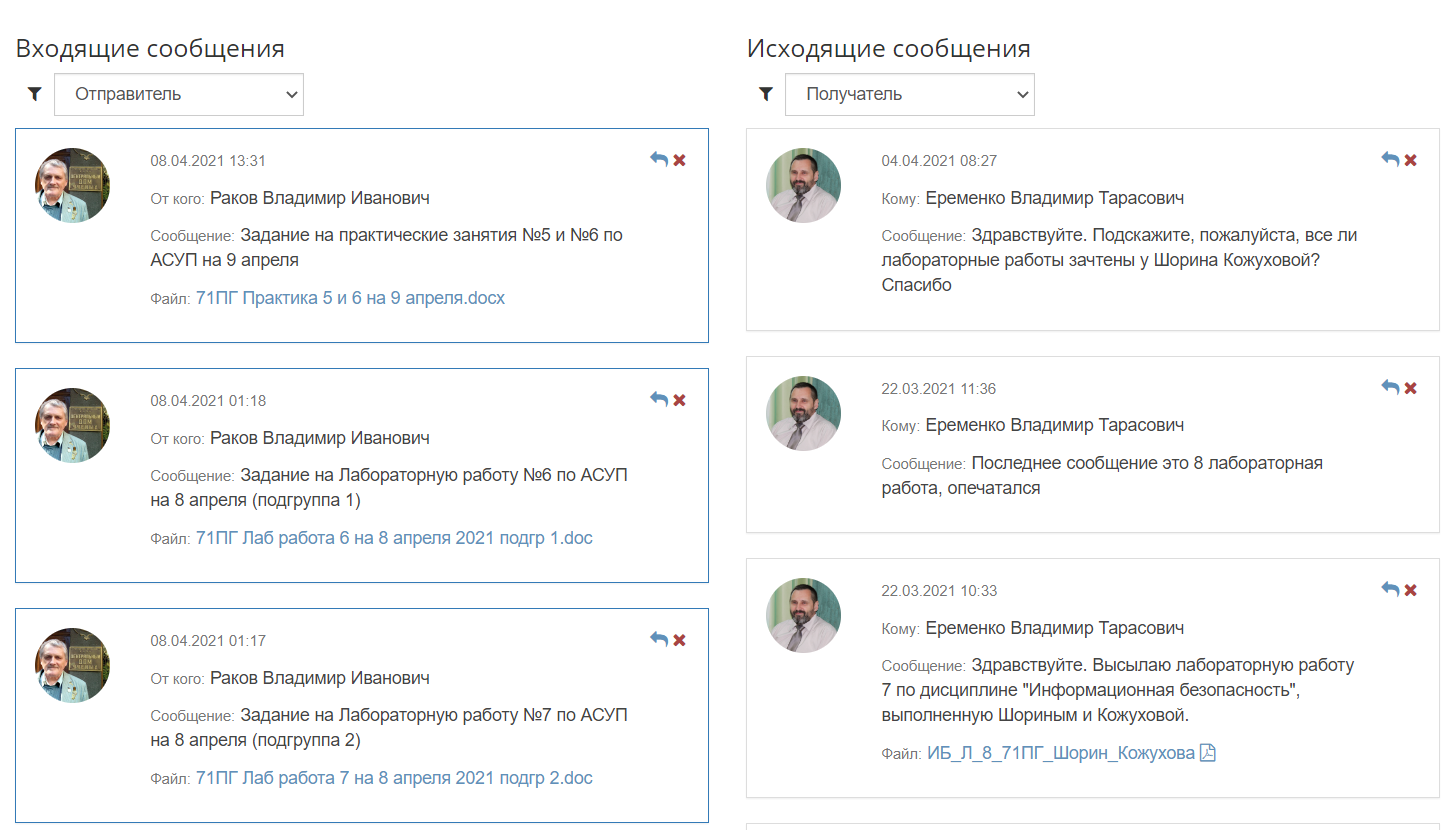


Рисунок 1.4 – Интерфейс веб-версии системы сообщений ОГУ им. И.С. Тургенева (продолжение)

Следующим предметом рассмотрения выберем популярный мессенджер «WhatsApp», который доступен на различных платформах (в том числе, и мобильных).

Основной функционал данного приложения заключается в следующем:

* возможность обмена текстовыми сообщениями между пользователями;
* возможность обмена аудио- и видеоматериалами, а также электронными документами;
* создание как индивидуальных, так и групповых чатов;
* аудио- и видеозвонки;
* добавление контактов с контактной книги телефона в приложение;

На рис. 1.5 представлен интерфейс мобильного приложения «WhatsApp», скачанного с мобильного сервиса «Google Play», на котором можно заметить достаточно удобный и интуитивно понятный интерфейс.

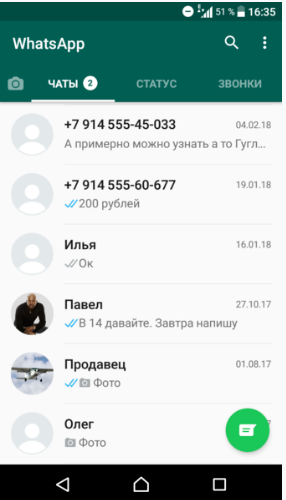


Рисунок 1.5 – Интерфейс приложения «WhatsApp»

Однако, в данном мессенджере отсутствует какая-либо возможность объединения чатов в группы по какому-либо критерию. Также, приложение направлено на предоставление услуг связи широкому кругу лиц, поэтому там отсутствует возможность взаимодействия с той или иной ЕОИС.

Основываясь на исследовании данных, доступных в открытых источниках сети Интернет, получены следующие сведения об архитектуре «WhatsApp»:

* языки программирования: Erlang, PHP;
* база данных: Mnesia;
* операционная система: FreeBSD;
* веб-серверы: Yaws, lighttpd;
* протокол обмена сообщениями: свой XMPP [1].

Следующим косвенным аналогом рассматривается «Viber». Данный мессенджер имеет аналогичный с предыдущим основной функционал. Отличия между ними заключаются на немного более глубоким уровне рассмотрения и выражаются в следующем:

* в «WhatsApp» присутствует веб-версия приложения, а в «Viber» нет;
* в «Viber» можно создавать опросы в групповом чате;
* возможность создания секретных чатов в «Viber» и т.д.

На рис. 1.6 представлен интерфейс мессенджера «Viber». Как можно заметить, все достаточно удобно и понятно: представлен список чатов, в которые можно перейти, просто выбрав нажатием необходимый, можно создать новый чат посредством нажатия на соответствующую иконку и т.д.

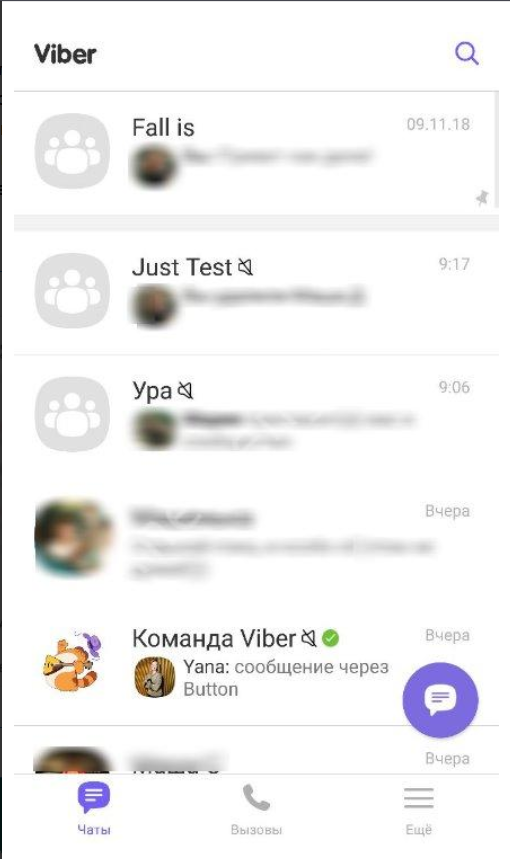


Рисунок 1.6 – Интерфейс приложения «Viber»

Однако, в мессенджере «Viber» также, как и в «WhatsApp», ожидаемо отсутствует возможность взаимодействия с ЕОИС высших учебных заведений, а также нет функции группировки чатов.

Проведя исследование доступной информации об архитектуре серверной части мессенджера «Viber», получился следующий стек технологий:

* + языки программирования: С++, Python, Java, Shell script, PHP;
  + база данных: Couchbase, несколько кластеров NoSQL баз данных;
  + операционная система: нет данных;
  + веб-сервер: nginx
  + протокол обмена сообщениями: WebRTC [2].

Архитектура всех представленных аналогов, также как и архитектура разрабатываемого сервиса обмена сообщениями, построены по принципу MVC. Архитектурный паттерн MVC (Model-View-Controller, Модель-Представление-Контроллер) представляет собой схему разделения создаваемого приложения, состоящую из трех частей:

* модель – предоставляет данные и осуществляет отклик на команды, приходящие от контроллера (при этом изменяет свое состояние);
* представление – реагирует на изменения в модели и отображает ее данные удобным для пользователя способом;
* контроллер – получает действия от пользователя и, используя модель и представление, реализует их.

Сервис, который разрабатывается в рамках данной работы имеет следующие преимущества относительно аналогов:

* взаимодействие с ЕОИС высшего учебного заведения (ОГУ им. И.С. Тургенева в данном случае);
* возможность группировки чатов по интересам (в данном случае, по дисциплинам, курсам, типам предмета и т.д.);
* отсутствие необходимости хранения информации о других участниках учебного процесса. Поскольку сервис взаимодействует с ЕОИС ВУЗа, то вся информация берется из запросов на сервер ЕОИС, благодаря чему у пользователя отсутствует необходимость хранения на своем устройстве большого количества телефонных номеров, контактных данных и остальной необходимой информации о других участниках).

Серверная подсистема системы обмена сообщениями будет разрабатываться с применением следующих технологий:

* язык программирования: PHP (с использованием фреймворка Laravel);
* база данных: Microsoft SQLServer;
* операционная система: Windows;
* веб-сервер: Apache;
* протокол обмена сообщениями: HTTP.

Изучив существующие аналоги, для упрощения процесса анализа преимуществ и недостатков каждого приложения необходимо построить сравнительную таблицу, в которой будут четко представлены результаты их изучения. В табл. 1.1 представлен сравнительный анализ аналогов разрабатываемого программного обеспечения.

Таблица 1.1 – Сравнение выбранных аналогов с разрабатываемым приложением

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий сравнения | WhatsApp | Viber | Web-версия ОГУ | Разрабатываемый сервис |
| возможность создания индивидуального чата | + | + | Представлено в виде выбора преподавателя/студента | + |
| возможность создания группового чата | + | + | Представлено в виде выбора группы, которой хочет написать преподаватель | + |

Продолжение таблицы 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| возможность объединения чатов в совокупности по критериям | - | - | - | + |
| возможность обмена материалами (аудио- и видеоформата, изображения, электронными документы и т.д.) | + | + | Документы, изображения и архивы | + |
| Критерий сравнения | WhatsApp | Viber | Web-версия ОГУ | Разрабатываемый сервис |
| простота интеграции в систему управления учебным процессом высшего учебного заведения (в данном случае, ОГУ им. И.С. Тургенева) | - | - | + | + |
| возможность автоматизированного создания чатов | - | - | Список доступных групп/преподавателей, исходя из нагрузки | + |

Продолжение таблицы 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| язык(и) программирования, фреймворк(и) | Erlang, PHP | С++, Python, Java, Shell script, PHP | PHP, Zend | PHP, Laravel |
| база данных | Mnesia | Couchbase, NoSQL | Microsoft SQLServer | Microsoft SQLServer |
| операционная система | FreeBSD | н/д | CentOS | Windows |
| веб-сервер | Yaws, lighttpd | Nginx | Apache | Apache |
| протокол обмена сообщениями | Свой XMPP | WebRTC | HTTP | HTPP |

На основе представленных в табл. 1.1 данных, можно сделать вывод о том, что все сервисы обмена сообщения помимо прямых функций по обмену сообщениями и созданию индивидуальных и групповых чатов, предоставляют также возможности по обмену электронными документами, аудио- и видеоматериалами. К тому же, каждый из аналогов имеет удобный и понятный интерфейс, что является одним из самых важных критериев популярности приложения. Однако ни одно из представленных решений не позволяет группировать чаты по критериям.

Отсюда можно сделать вывод о том, что на текущий момент на рынке программного обеспечения отсутствует открытое решение, которое поможет максимально упростить процесс взаимодействия студента и преподавателя. Исходя из проведенного анализа выявленных аналогов можно сделать вывод об актуальности разрабатываемого сервиса обмена сообщениями.

## Постановка задачи и формулировка требований

Изучив структуры сервисов личных кабинетов студента и преподавателя, а также проведя анализ функциональных возможностей существующей системы обмена сообщениями, мы можем сделать вывод о том, что имеющаяся в данный момент система имеет некоторое количество недостатков, а именно:

1. сложный процесс доступа к сервису – необходимо сначала авторизироваться в личном кабинете, затем выбрать соответствующий пункт меню и затем уже имеется возможность отправки сообщения. Помимо этого, также накладывается ограничение на продолжительность сессии авторизации, что влечет за собой периодические повторения вышеуказанных действий для повторного доступа к сервису обмена сообщениями;
2. неудобный процесс формирования и отправки сообщения – для студента и преподавателя: необходимо сначала выбрать преподавателя из списка, затем набрать сообщение в соответствующей поле и прикрепить при необходимости файл с дополнительными материалами;
3. отсутствие уведомлений о сообщениях – если участнику образовательного процесса приходит новое сообщение, то он никаким образом не сможет об этом узнать, кроме личной проверки системы сообщений (что включает также первый пункт о сложности процесса доступа);
4. неудобный интерфейс отображения сообщений – все сообщения отражаются в том порядке, в котором были получены/отправлены на нескольких страницах;
5. отсутствие какой-либо группировки сообщений по темам – все сообщения можно отфильтровать только по их отправителю или получателю, что накладывает определенные трудности по нахождению какого-либо определенного сообщения по интересующей теме в случае наличия длинной переписки с конкретным пользователем;

На основе выявленных недостатков, а также проведенного анализа выбранных аналогов можем сделать вывод о том, что главной задачей данной работы является разработка сервиса обмена сообщениями, который упростит процесс обмена сообщениями между преподавателем и студентом, т.е. позволит им мгновенно обмениваться сообщениями, получать уведомления о получении нового сообщения, группировать чаты по тематикам и т.д.

Разрабатываемый сервис обмена сообщениями будет состоять из серверной части и мобильного приложения. Мобильное приложение соответствует представлению в архитектурном паттерне MVC, а сервер содержит в себе модель и контроллер, соответственно.

## Функциональные возможности сервиса обмена сообщениями

Разрабатываемый сервис обмена сообщениями должен реализовывать следующие возможности:

* создание индивидуальных и групповых чатов;
* группировка чатов по типам занятий (лабораторная, практика и т.д.);
* просмотр вложенных документов в текущем чате;
* возможность отправки в чат электронных документов различных типов;
* отображение количества непрочитанных сообщений в каждом чате;
* управление сообщениями (удаление, редактирование);
* управление чатами администратором подсистемы управления чатами.

## Нефункциональные требования к серверной подсистеме

К нефункциональным требованиям можно отнести:

* целевая платформа - Windows;
* язык программирования – PHP;
* фреймворк – Laravel;
* СУБД – Microsoft SQL Server;
* протокол передачи данных – HTTP;
* архитектурный стиль взаимодействия с серверной подсистемой – REST API;
* веб-сервер – Apache.

# ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ К СЕРВИСУ ОБМЕНА СООБЩЕНИЯМИ

## Разработка архитектуры

Наиболее распространенной архитектурой для построения веб-приложений является клиент-серверная архитектура. Оба компонента представляют собой программное обеспечение, в большинстве случаев расположенные на разных вычислительных машинах, которые взаимодействуют между собой используя различные сетевые протоколы, но также они могут находиться и на одном устройстве. Клиентские программы являются потребителями информации и делают соответствующие запросы на сервер. Серверные программы, в свою очередь, являются поставщиками данной информации и предоставляют клиентам данные в соответствии с полученным запросом [3].

Применяемый для разработки серверной подсистемы веб-фреймворк Laravel предназначен для разработки приложений с использованием архитектурной модели MVC (рис. 2.1).

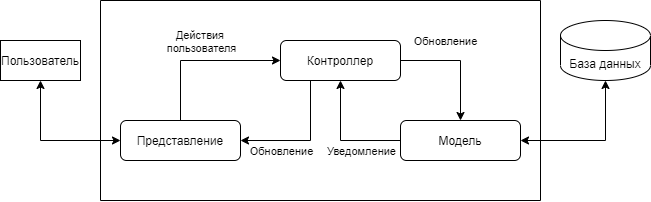


Рисунок 2.1 – Схема архитектуры MVC

Сама серверная подсистема сервиса обмена сообщениями будет представлять собой контроллер и модель, а в качестве представления будет выступать параллельно разрабатываемое мобильное приложение Взаимодействие между представлением (мобильным приложением) и контроллером и моделью (сервером) будет осуществляться с применением API. Для реализации данного взаимодействия в веб-фреймворке Laravel предусмотрены отдельные контроллеры – REST-контроллеры, представляющие собой дополнительный слой разделения логики обработки GET- и POST-запросов.

Представление (мобильное приложение), в зависимости от действий пользователя, делает запрос на сервер по соответствующему API (application programming interface, программный интерфейс приложения). Программный интерфейс приложения исходя из полученной команды выбирает необходимый контроллер. Контроллер использует связанную с ним модель (модели) для выполнения поставленной задачи и обновляет ее данные. Модель, в свою очередь, производит все необходимые действия над своими данными (производит запросы в базу данных, проверяет их на корректность и т.д.), уведомляет контроллер о результатах произведенных действий и возвращает измененные данные. Контроллер производит свои действия над полученными от модели данными при необходимости и передает их в качестве ответа представлению, которое посредством своего независимого функционала визуализирует полученные результаты.

Именно использование API и позволяет делать клиент независимо от сервера. Архитектурным стилем, применяемым при разработке API, будет являться REST-API.

## Разработка функциональных спецификаций

Для упрощения процесса разработки и описания функциональных спецификаций сервиса обмена сообщениями можно воспользоваться диаграммой вариантов использования (use case diagram) UML. Данная диаграмма необходима для того, чтобы описать отношения между пользователями и возможностями моделируемой системы или части ее возможной функциональности («актерами» и «прецендентами», соответственно, в контексте языка UML).

Диаграмма всех вариантов использования разрабатываемого сервиса обмена сообщениями представлена на рис. 2.2.

Подробное описание каждого варианта использования представлено в табл. 2.1 – 2.7.

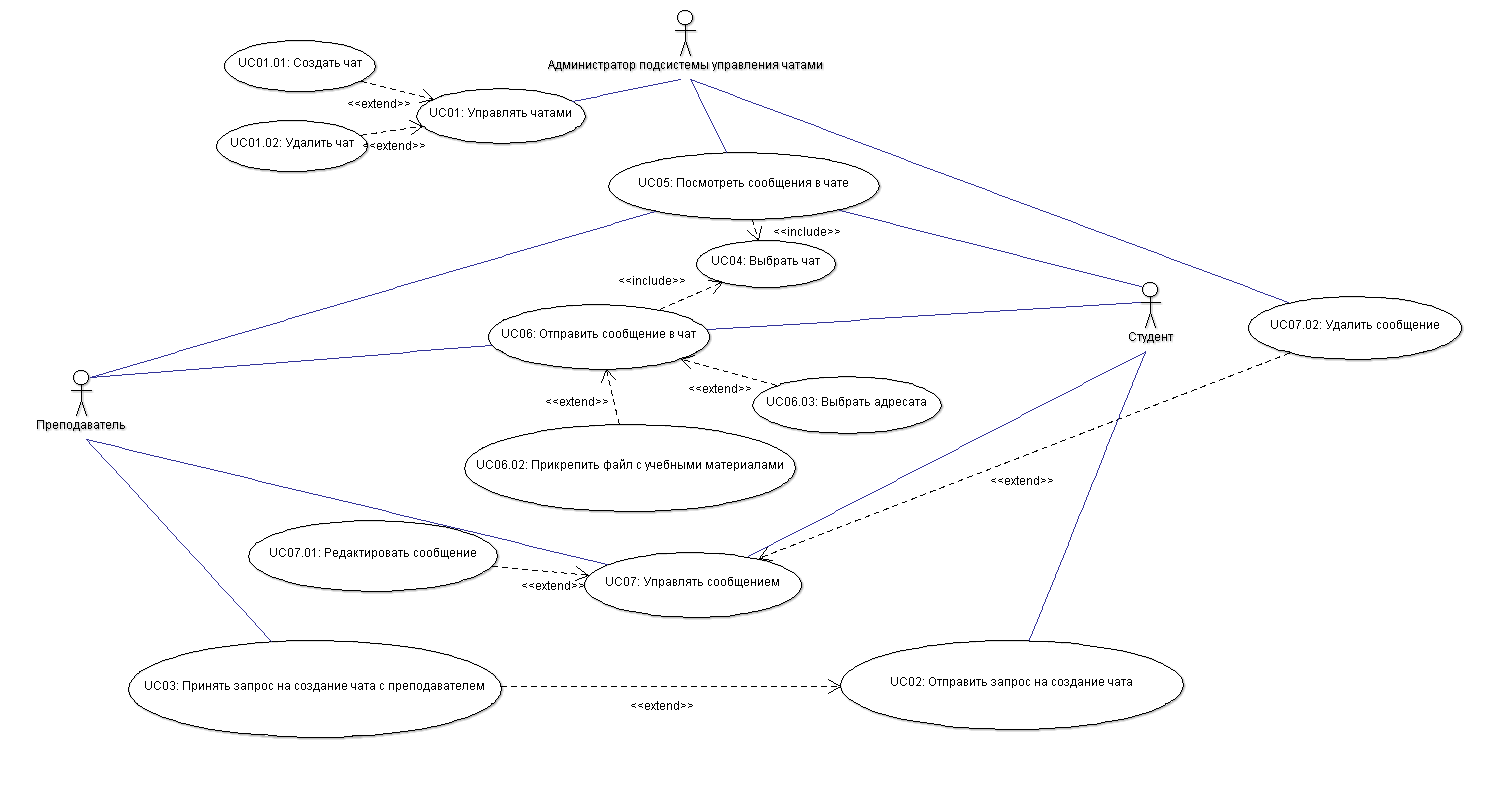


Рисунок 2.2 – Диаграмма вариантов использования приложения

Таблица 2.1 – UC01: Управлять чатами

|  |  |
| --- | --- |
| **Вариант использования** | **UC01: Управлять чатами** |
| Описание (основной поток) | 1. Система выводит список существующих чатов в режиме управления 2. Администратор выбирает чат для управления 3. Администратор выбирает функцию «Управлять чатом» 4. Система выводит сведения о чате из БД 5. Администратор добавляет / удаляет пользователей в чат и выбирает одно из действий:   5а: Сохранить изменения: система сохраняет изменения в БД, добавляет / удаляет чат в списках чатов пользователей. Возврат к п.1  5б: Отмена: выводится сообщение для подтверждения действия; при подтверждении изменения не сохраняются. Возврат к п.1 |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Триггер | Выбрана функция «Управлять чатами» |
| Предусловия | PRE-1: Авторизоваться в системе в роли администратора подсистемы управления чатами |
| Постусловия | POST-1: Измененный список чатов сохранен в БД  POST-2: В списке чатов пользователя отображаются актуальные чаты |
| Альтернативные потоки | A1: Удаление чата:   1. Шаг 1 основного потока 2. Шаг 2 основного потока 3. Администратор выбирает функцию «Удалить чат» 4. Выводится сообщение для подтверждения действия 5. Администратор выбирает одно из действий:   5а: Удалить чат: система удаляет чат из списков чатов пользователей, очищает сведения о выбранном чате в БД и сохраняет изменения в БД. Возврат к п.1  5б: Отмена: выводится сообщение для подтверждения действия; при подтверждении изменения не сохраняются. Возврат к п.1  А2: Создание чата:   1. Шаг 1 основного потока 2. Администратор выбирает функцию «Создать чат» 3. Система выводит список пользователей из БД для создания чата 4. Администратор выбирает пользователей из списка и создает новый чат 5. Система добавляет чат в список чатов, добавляет чат в чаты выбранных пользователей, сохраняет изменения в БД. Возврат к п.1 |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Исключения | Нет |

Таблица 2.2 **–** UC02: Отправить запрос на создание чата

|  |  |
| --- | --- |
| **Вариант использования** | **UC02: Отправить запрос на создание чата** |
| Описание (основной поток) | 1. Система отображает список чатов студента 2. Студент выбирает функцию «Создать чат с преподавателем» 3. Система выводит список преподавателей из БД 4. Студент выбирает преподавателя из списка для создания чата 5. Система отправляет преподавателю запрос на создание чата со студентом. Возврат к п.1 |
| Триггер | Выбрана функция «Отправить запрос на создание чата» |
| Предусловия | PRE-1: Пройдена аутентификация |
| Постусловия | POST-1: В списке чатов преподавателя отображается запрос на создание чата со студентом |
| Альтернативные потоки | Нет |
| Исключения | Нет |

Таблица 2.3 – UC03: Принять запрос на создание чата с преподавателем

|  |  |
| --- | --- |
| **Вариант использования** | **UC03: Принять запрос на создание чата с преподавателем** |
| Описание (основной поток) | 1. Преподаватель выбирает раздел «Запросы на создание чатов» 2. Система отображает существующие запросы |

Продолжение таблицы 2.3

|  |  |
| --- | --- |
|  | 1. Преподаватель отвечает на запрос 2. Система создает чат, добавляет чат в список чатов, добавляет чат в чаты выбранных пользователей, сохраняет изменения в БД. Возврат к п.1 |
| Триггер | Выбрана функция «Принять запрос на создание чата с преподавателем» |
| Предусловия | PRE-1: Пройдена аутентификация |
| Постусловия | POST-1: Измененный список чатов сохранен в БД  POST-2: В списке чатов пользователя отображаются актуальные чаты |
| Альтернативные потоки | Нет |
| Исключения | Нет |

Таблица 2.4 – UC04: Выбрать чат

|  |  |
| --- | --- |
| **Вариант использования** | **UC04: Выбрать чат** |
| Описание (основной поток) | 1. Система отображает список чатов пользователя 2. Пользователь выбирает чат для работы |
| Триггер | Доступно на главном экране приложения |
| Предусловия | PRE-1: Пройдена аутентификация |
| Постусловия | POST-1: Отображение материалов выбранного чата на экран |
| Альтернативные потоки | Нет |
| Исключения | Нет |

Таблица 2.5 – UC05: Посмотреть сообщения в чате

|  |  |
| --- | --- |
| **Вариант использования** | **UC05: Посмотреть сообщения в чате** |
| Описание (основной поток) | 1. Система отображает сообщения чата с разделением на прочитанные и непрочитанные 2. Пользователей просматривает сообщения чата |
| Триггер | Выбрана функция «Посмотреть сообщения в чате» |
| Предусловия | PRE-1: Пройдена аутентификация  PRE-2: Выбран чат |
| Постусловия | POST-1: Непрочитанные сообщения отображаются как прочитанные |
| Альтернативные потоки | Нет |
| Исключения | Нет |

Таблица 2.6 – UC06: Отправить сообщение в чат

|  |  |
| --- | --- |
| **Вариант использования** | **UC06: Отправить сообщение в чат** |
| Описание (основной поток) | 1. Система отображает экранную клавиатуру |
|  | 1. Пользователь вводит текст сообщения и выбирает оно из действий:   3а: Отправить сообщение: система отправляет сообщение в чат, сохраняет сообщение в БД. Возврат к чату  3б: Отмена: система очищает окно ввода сообщения. Изменения не сохраняются. Возврат к чату |

Продолжение таблицы 2.6

|  |  |
| --- | --- |
| Триггер | Пользователь выбирал поле ввода сообщения в качестве текущего элемента управления |
| Предусловия | PRE-1: Пройдена аутентификация  PRE-2: Выбран чат |
| Постусловия | POST-1: Обновленный чат сохранен в БД  POST-2: В чате пользователям отображаются непрочитанные сообщения |
| Альтернативные потоки | А1: Прикрепить файл с учебными материалами   1. Шаг 1 основного потока 2. Шаг 2 основного потока 3. Шаг 3 основного потока (до выбора действия) 4. Пользователь выбирает функцию «Прикрепить файл с учебными материалами» 5. Система отображает список возможных типов файлов для вложения, открывает файловую систему устройства для выбора файла 6. Пользователь выбирает файл и выбирает одно из действий:   6а: Прикрепить файл: система прикрепляет файл к отправляемому сообщению, сохраняет файл в БД, закрывает файловую систему устройства. Возврат к п.3 основного потока  6б: Отмена: система закрывает файловую систему устройства, файл не сохраняется в БД. Возврат к п.3 основного потока  А3: Выбрать адресата:   1. Шаг 1 основного потока 2. Шаг 2 основного потока |

Продолжение таблицы 2.6

|  |  |
| --- | --- |
|  | 1. Шаг 3 основного потока (до выбора действия) 2. Пользователь выбирает функцию «Отправить сообщение адресату» 3. Система отображает список пользователей чата 4. Пользователь выбирает адресата и одно из действий:   6а: Отправить сообщение: система отправляет сообщение в выбранному пользователю, сохраняет сообщение в БД. Возврат к чату  3б: Отмена: система очищает окно ввода сообщения и выбора адресата. Изменения не сохраняются. Возврат к чату |
| Исключения | Нет |

Таблица 2.7 – UC07: Управлять сообщением

|  |  |
| --- | --- |
| **Вариант использования** | **UC07: Управлять сообщением** |
| Описание (основной поток) | 1. Пользователь выбирает функцию «Редактировать сообщение» 2. Система отображает сообщение в окне редактирования 3. Пользователь вводит изменения в сообщение и выбирает одно из действий:   3а: Сохранить изменения: система сохраняет изменения в сообщении в БД. Возврат к чату  3б: Отмена: выводится сообщение для подтверждения действия; при подтверждении изменения не сохраняются. Возврат к чату |
| Триггер | Выбрана функция «Управлять сообщением» |

Продолжение таблицы 2.7

|  |  |
| --- | --- |
| Предусловия | PRE-1: Пройдена аутентификация  PRE-2: Выбран чат, имеющий отправленные сообщения |
| Постусловия | POST-1: Отредактированный чат сохранен в БД  POST-2: В чате пользователям отображаются актуальные сообщения |
| Альтернативные потоки | А1: Удалить сообщение:   1. Пользователь выбирает функцию «Удалить сообщение» 2. Выводится сообщение для подтверждения действия 3. Пользователь выбирает одно из действий:   3а: Удалить сообщение: система отправляет запрос на удаление сообщения из БД администратору, удаляет сообщение из чата, обновляет чат. Возврат к чату  3б: Отмена: выводится сообщение для подтверждения действия; при подтверждении изменения не сохраняются. Возврат к чату |
| Исключения | Нет |

## Разработка информационных спецификаций серверной подсистемы сервиса обмена сообщениями

Для разработки информационных спецификаций воспользуемся контекстной диаграммой классов (или диаграммой классов анализа, содержащей абстракции предметной области). Данная диаграмма является более абстрактным представлением классической диаграммы классов.

Для того, чтобы разработать контекстную диаграмму классов, выделим элементы системы, взаимодействие между которыми необходимо отобразить на ней. В первую очередь, это должны быть база данных, клиентские приложения (или просто клиенты) и API. Также, на стадии анализа мы можем выделить некоторые контроллеры, которые будут осуществлять управление запросами, и модели данных, которые будут связаны с базой данных.

На основании вышеизложенной информации на рис. 2.3 представлена контекстная диаграмма классов для серверной подсистемы сервиса обмена сообщениями.

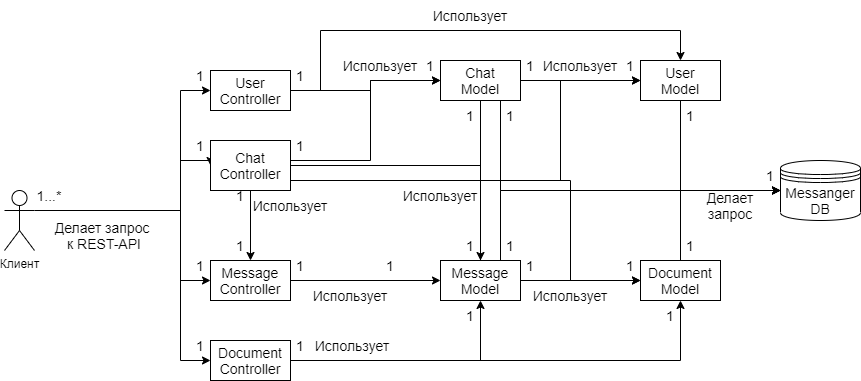


Рисунок 2.3 – Контекстная диаграмма классов

Суть данной диаграммы состоит в следующем: есть некоторое количество клиентов, которые делают запросы к документированному API сервера. Каждый запрос связан с определенной конечной точкой в API, которая, в свою очередь, связана с соответствующим контроллером. Контроллер обрабатывает пришедшую команду и, на основе результатов обработки, обращается к необходимой модели данных. Модель данных связана с определенной таблицей (таблицами) в базе данных, куда делает соответствующий запрос на выборку информации, необходимой клиенту в зависимости от его запроса.

## Логическая модель API

Применительно к разрабатываемому сервису обмена сообщениями, архитектурный паттерн MVC будет дополнен применением API ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Application Programming Interface, программный интерфейс приложения). API, по своей сути, является неким «контрактом», который заключается между разными приложениями или между частями одного и того же приложения. Применительно к концепции клиент-сервер, API чаще всего реализуется на стороне сервера, а клиент уже обращается к нему, вызывая соответствующие функции, то есть делает запросы на сервер.

Данный контракт определяет то, каким образом стороны контракта могут взаимодействовать друг с другом, то есть:

* какие функции доступны для использования клиентом и их описание;
* какие данные принимает сервер от клиента и в каком формате;
* какие данные отдает сервер клиенту и в каком формате;

Существует два вида разработки API: CodeFirst и ContractFirst. Первый подход – CodeFirst – состоит в том, что сначала происходит разработка кода сервера и только после того, как его функционал готов (в том числе и API, по которому будет возможно к нему обратиться), формируется документация по данному API.

Преимущества данного подхода:

* минимальные усилия по генерации контракта, поскольку API уже написано и его остается только задокументировать (на данный момент существуют средства, позволяющие автоматизировать этот процесс);
* актуальность контракта, поскольку он генерируется из кода.

Недостатки данного подхода:

* + отсутствие возможности параллельной разработки. Из-за того, что сначала необходимо разработать сервис, а затем сгенерировать контракт, разработка клиентской части откладывается на неопределенное время до получения контракта. Невозможно нормально разработать потребителя не имея договора.
  + нет общего направления работы команд. Этот недостаток вытекает из предыдущего пункта и означает, что до момента получения контракта, заинтересованные стороны, разрабатывающие клиентскую и серверную части, могут по-разному представлять себе как конечные результаты работы, так и функционал приложения [4].

Суть второго подхода – ContractFirst – сводится к тому, что сначала определяется контракт, по которому будут общаться стороны, и только после этого команды могут приступать к разработке клиента и сервера соответственно.

Преимущества данного подхода:

* возможность параллельной разработки. Исходя из того, что контракт разработан, команды могут спокойно приступать к разработке своих частей;
* точное представление об ожиданиях обеих сторон. Если функционал обеих сторон разрабатывается в разных темпах, то для проведения его тестирования нет необходимости ждать другую сторону. Поскольку контракт уже готов, то всем известно, что и в каком виде отправлять и получать, поэтому можно использовать заглушки.

Недостатки данного подхода:

* наличие дополнительных затрат в начале разработки. Необходимо потратить большое количество времени и сил, чтобы определиться с контрактом, чтобы он был максимально четким и соответствовал поставленным задачам. При этом, он не должен будет часто меняться.
* сложность обновления контракта. Изменение контракта влияет на все заинтересованные стороны. Поэтому, необходимо иметь механизм, который поможет упростить процесс оповещения и внедрения изменений разработчиками [5].

В данной работе будет использоваться второй подход (контракт в первую очередь), поскольку разработка клиентской и серверной частей ведется параллельно разными разработчиками.

Также, будет применяться архитектурный стиль REST API. Его суть состоит в следующем: клиент вызывает удаленную функцию, которая представляет собой обычный HTTP-запрос (GET или POST), передавая данные в качестве параметров запроса. Каждому ресурсу на сервере назначается свой идентификатор URI (универсальный идентификатор ресурса). Например, если мы хотим получить всех пользователей – GET/users. Также, в запросе указывается метод, описывающий действие, которое необходимо произвести с данными:

* GET – получить информацию о ресурсе;
* POST – создать новый ресурс;
* PUT(UPDATE) – обновить существующий ресурс;
* DELETE – удалить ресурс.

Кроме того, немаловажной частью являются и метаданные, возвращаемые в ответе сервером. К ним могут относиться: сообщения об ошибках, различные свойства передаваемого ресурса (например, общее число записей) и т.д.

Для каждой функции API, которая будет у сервиса, необходимо определить метод работы с данными, URI, данные, получаемые при запросе и отправляемые в ответе, формат этих данных (схемы данных, в которых они будут передаваться), метаданные.

При проектировании API, в первую очередь, необходимо определиться с ресурсами, которые будет принимать и отдавать сервер. Для нашей задачи на основе контекстной диаграммы классов, представленной в п. 2.3, получаем следующий перечень ресурсов, структуру которых необходимо определить:

* пользователь;
* чат;
* сообщение;
* информация о документе;

На рис. 2.4 – 2.7 представлены разработанные схемы для представленных выше ресурсов.

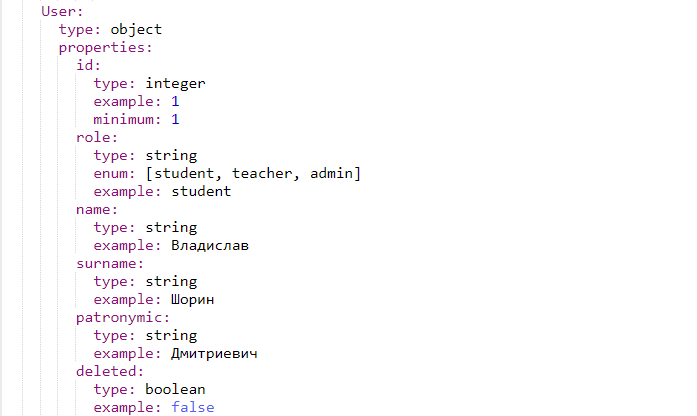


Рисунок 2.4 – Схема ресурса «Пользователь»

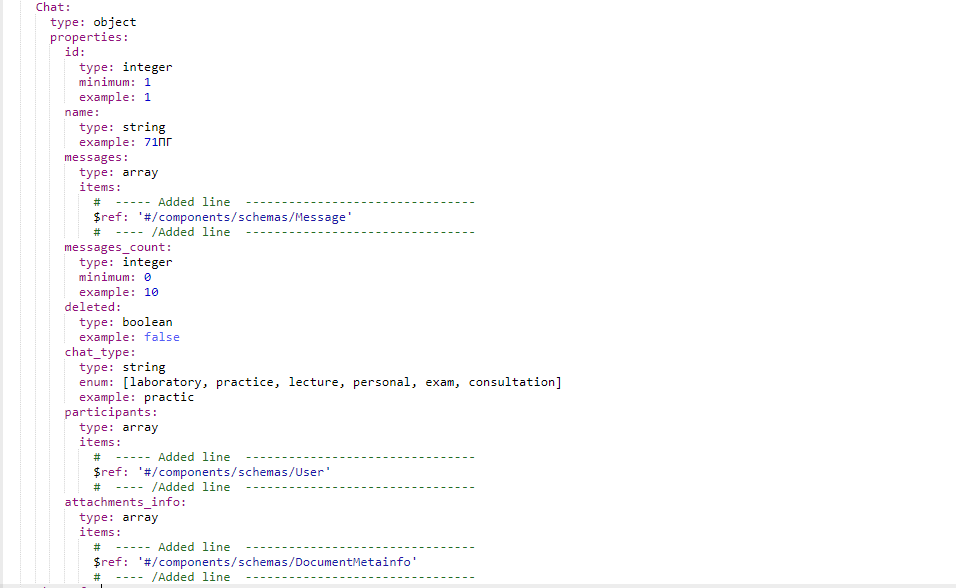


Рисунок 2.5 – Схема ресурса «Чат»

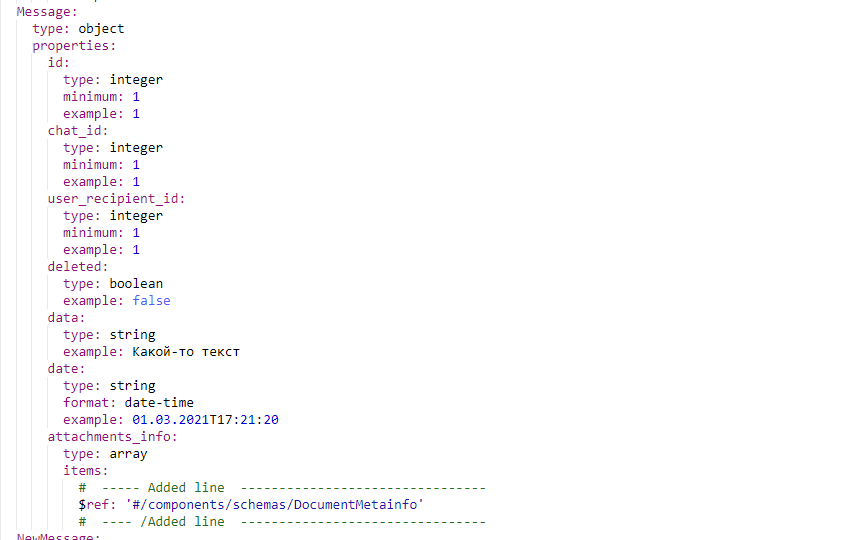


Рисунок 2.6 – Схема ресурса «Сообщение»

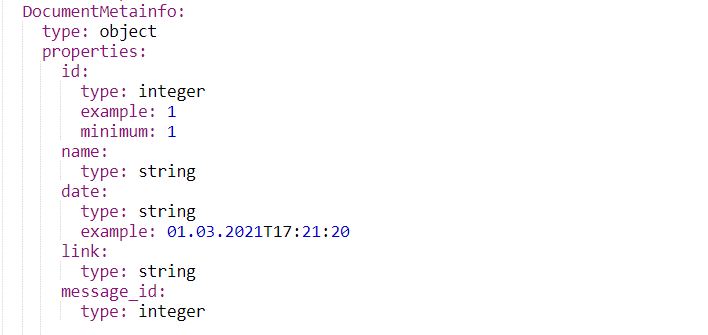


Рисунок 2.7 – Схема ресурса «Информация о документе»

# ПРОЕКТИРОВАНИЕ СЕРВЕРНОЙ ПОДСИСТЕМЫ СЕРВИСА ОБМЕНА СООБЩЕНИЯМИ

## Проектирование структуры программного обеспечения

Следующим этапом разработки серверной подсистемы сервиса обмена сообщениями является разработка диаграммы классов.

На рис. 3.1 представлена спроектированная диаграмма классов.

Левая часть данной диаграммы соответствует набору классов-контроллеров, которые осуществляют управление процессом работы серверной подсистемы сервиса обмена сообщениями. Каждый класс-контроллер является расширением одного общего базового класса «Controller», наследуя некоторый общий функционал и методы работы с данными. Но при этом реализация общего функционала отличается и уникальна в рамках каждого класса.

Каждый класс-контроллер, в свою очередь, связан с одним или несколькими классами-моделями. Каждый класс-модель является расширением одного общего базового класса «Model», который предоставляет возможность взаимодействия с базой данных. Также, каждый класс-модель связан с соответствующей таблицей в базе данных, что значительно облегчает процесс взаимодействия с ней.

## Проектирование основных компонентов программного обеспечения

Рассмотрим более подробно некоторые классы, представленные на рис. 3.1.

В первую очередь, обратим внимание на основной класс-контроллер «Controller». Данный класс является базовым для всех контроллеров, используемых в работе программной системы, и содержит достаточно большую часть общего функционала, необходимого для реализации проектируемого функционала.

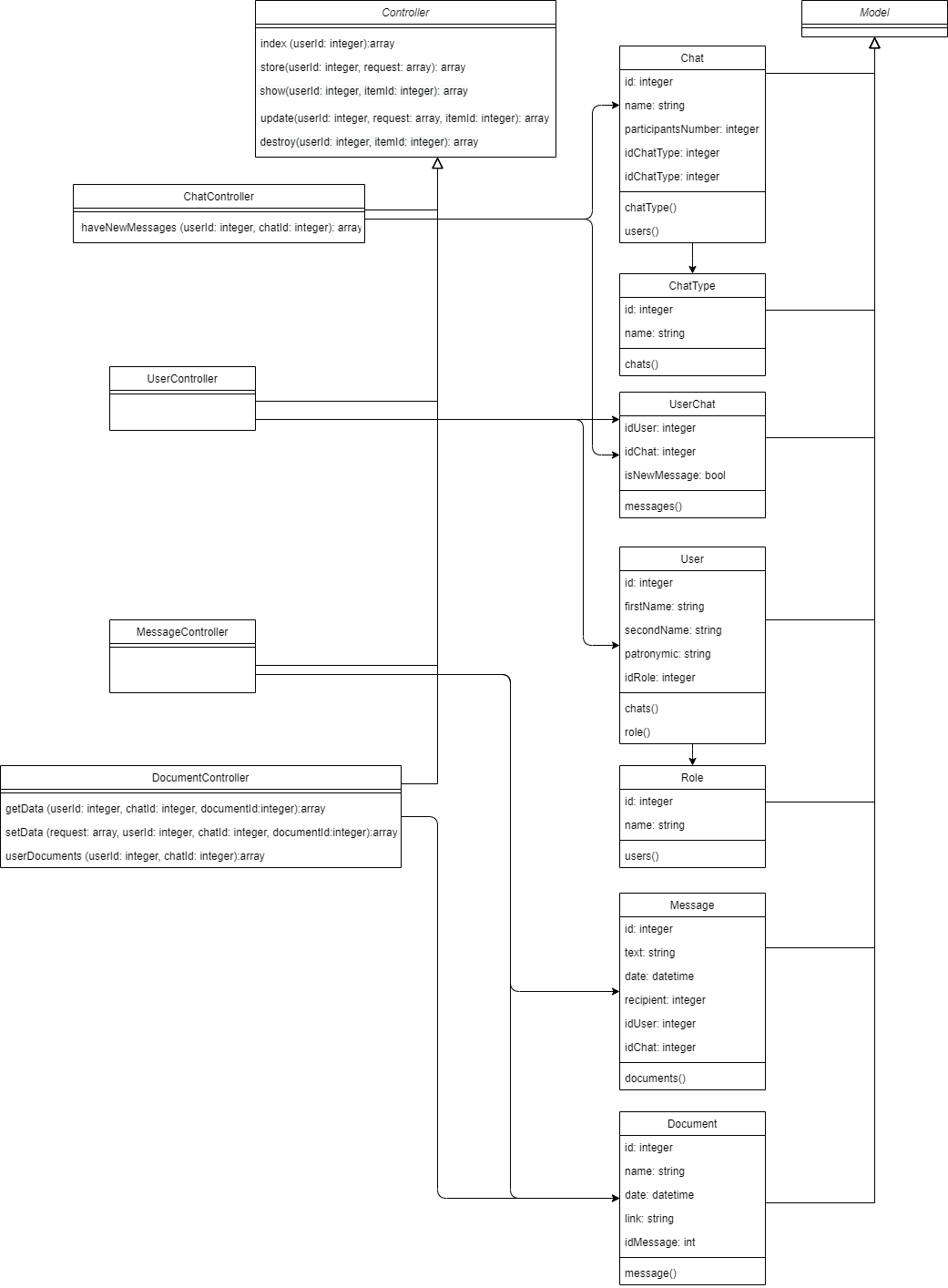


Рисунок 3.1 – Диаграмма классов

Каждый производный класс наследует следующие методы:

* index (userId: integer): array – получение списка значений экземпляров ресурса (например, список чатов);
* store (userId: integer, request: array): array – создание нового экземпляра ресурса (например, создание нового сообщения);
* show (userId: integer, itemId: integer): array – получение информации о каком-то конкретном экземпляре ресурса (например, получение информации о конкретном чате);
* update (userId: integer, request: array, itemId: integer): array – изменение информации о каком-либо конкретном экземпляре ресурса (например, изменение информации о документе);
* destroy (userId: integer, itemId: integer): array – удаление экземпляра ресурса (например, удаление сообщения).

Все производные классы не просто наследуют базовые методы, а, в большинстве случаев, переопределяют их, т.е. изменяют входные параметры, логику работы и возвращаемые значения в зависимости от своих нужд. К примеру, контроллер управления сообщениями MessageController в методе show может иметь следующие параметры: integer userId, integer chatId, integer messageId.

Изучим немного подробнее класс-наследник DocumentController. Данный класс служит для организации процесса работы с документами и связан с соответствующими маршрутами API. Он имеет помимо базовых также и некоторые индивидуальные методы:

* getData (userId: integer, chatId: integer, documentId: integer): array – при вызове позволяет пользователю загрузить с сервера необходимый документ;
* setData (request: array, userId: integer, chatId: integer, documentId: integer): array – при вызове позволяет пользователю загрузить новый документ на сервер;
* userDocuments (userId: integer, chatId: integer): array – при вызове позволяет получить список всех документов текущего пользователя в конкретном чате.

Производные классы UserController и MessageController не имеют каких-либо индивидуальных методов, поскольку базовых вполне достаточно для организации процесса управления ресурсами пользователя и чата соответственно.

Производный класс ChatController имеет один дополнительный метод haveNewMessages (int userId, int chatId), который позволяет проверить, есть ли новые сообщения в чате.

Каждый производный класс-модель имеет список полей, которые соответствуют столбцам базы данных, что позволяет удобно и быстро работать с ней. Методы позволяют без лишних усилий получить информацию модели, связанной с текущей моделью. К примеру, метод role() подкласса User позволяет быстро получить роль конкретного пользователя по ее номеру в базе данных.

## Проектирование API

Для получения доступа к ресурсам, описанным в пункте 2.5, необходимо определить «эндпоинты» (от англ. Endpoint – конечная точка) – тот адрес, на который посылаются сообщения от клиента. Они выражаются в URL, для каждого из которых определяется и описывается метод работы (тип метода, его описание, тело запроса и т.д.), а для каждого метода также описывается код и структура получаемого ответа. Основными URL будут являться:

* /api/v1/{userId}/requestToCreateChat – отправка запроса к администратору системы на создание нового чата с каким-то пользователем;
* /api/v1/{userId}/chats– получение информации обо всех чатах, доступных текущему пользователю, а также добавление нового чата;
* /api/v1/{userId}/chats/{chatId} – получение информации о конкретном чате, ее изменение и удаление данного чата;
* /api/v1/{userId}/chats/{chatId}/haveNewMessages – получение информации о наличии новых сообщений в конкретном чате пользователя;
* /api/v1/{userId}/chats/{chatId}/messages – получение списка сообщений и добавление нового сообщения в чат;
* /api/v1/{userId}/chats/{chatId}/messages/{messageId} – получение экземпляра сообщения, изменение его информации и его удаление;
* /api/v1/{userId}/chats/{chatId}/users – получение информации обо всех пользователях чата, а также добавление нового пользователя в чат;
* /api/v1/{userId}/chats/{chatId}/users/{userInChatId} – получение информации о пользователе, ее изменение и удаление
* /api/v1/{userId}/chats/{chatId}/documents– получение всех документов, вложенных в чат, а также добавление новых метаданных о документе;
* /api/v1/{userId}/chats/{chatId}/documents/{documentId} – получение метаданных о конкретном документе чата, а также ее изменение;
* /api/v1/{userId}/chats/{chatId}/documents/{documentId}/data– получение конкретного документа в чате и загрузка его на сервер;
* /api/v1/{userId}/chats/{chatId}/userDocuments – получение всех документов пользователя в чате.

В данной работе первый маршрут, отвечающий за создание запросов к администратору управления чатами на создание нового чата, не реализовывается и будет представлять из себя заглушку. Она состоит из пути и практически пустого метода, который возвращает ошибку, информирующую о том, что данный функционал еще не реализован. Это делается по той причине, что разрабатываемое в данной работе программное обеспечение является опытным прототипом и реализует только основной функционал для предварительного тестирования сервиса обмена сообщениями. Эта возможность будет реализовываться после успешного прохождения опытной эксплуатации и интеграции серверной подсистемы сервиса в ЕОИС ОГУ им. Тургенева.

На рис. 3.2 представлена структура и описание URL на примере получения списка чатов (/api/v1/{userId}/chats). Полная информация о данной конечной точке, а также об остальных представлена в Приложении 1.

Документирование API производится при помощи сервиса SwaggerHub, который использует фреймворк Swagger для спецификации RESTful API. Данный сервис позволяет удобно и интерактивно просматривать, и редактировать спецификацию, а также отправлять запросы для тестирования разработанных маршрутов.

Форматом, посредством которого будет производиться обмен данными между клиентами и сервером, будет JSON. JSON – текстовый формат обмена данными, основанный на JavaScript. Он достаточно легок для чтения человеком и совместим практически со всеми языками программирования. Помимо этого, для многих языков программирования существует уже готовый код для создания и обработки данных в формате JSON.

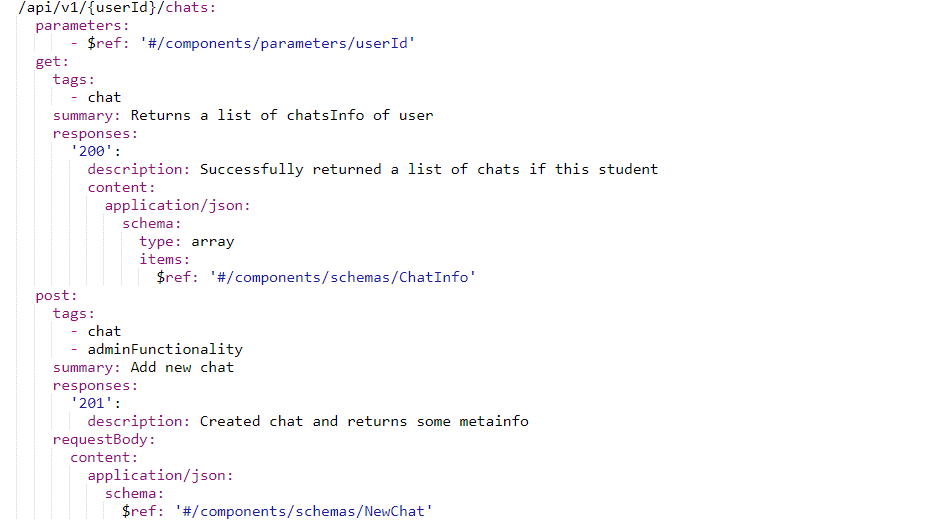


Рисунок 3.2 – Структура и описание URL получения списка чатов

## Проектирование базы данных

Для хранения всей информации, использующейся в процессе работы сервиса обмена сообщениями, необходимо также спроектировать базу данных.

На основе структуры программного обеспечения, представленного в п. 3.1. выделим следующие основные сущности, между которыми необходимо организовать взаимосвязь: пользователь, чат, сообщение, документ.

На рис. 3.3 представлена логическая схема базы данных, спроектированная на основе анализа предметной области.

Выделенные цветом таблицы будут использоваться в процессе интеграции разрабатываемого программного продукта с ИСУУП. Таблица «Пользователь» будет использоваться для получения дополнительной информации об участниках учебного процесса (студенты, преподаватели и т.д.), а таблица «Чат» для получения необходимой информации о дисциплинах, для которых создается чат.

Ключевыми сущностями на данной схеме являются:

* + Пользователь;
  + Роль;
  + Пользователь\_Чат;
  + Сообщение;
  + Документ;
  + Чат;
  + Тип чата.

Таблица «Пользователь» имеет следующие поля: Код (первичный ключ таблицы), Имя, Фамилия, Отчество, Удаленный, Роль\_Код (внешний ключ по отношению к таблице «Роль»).

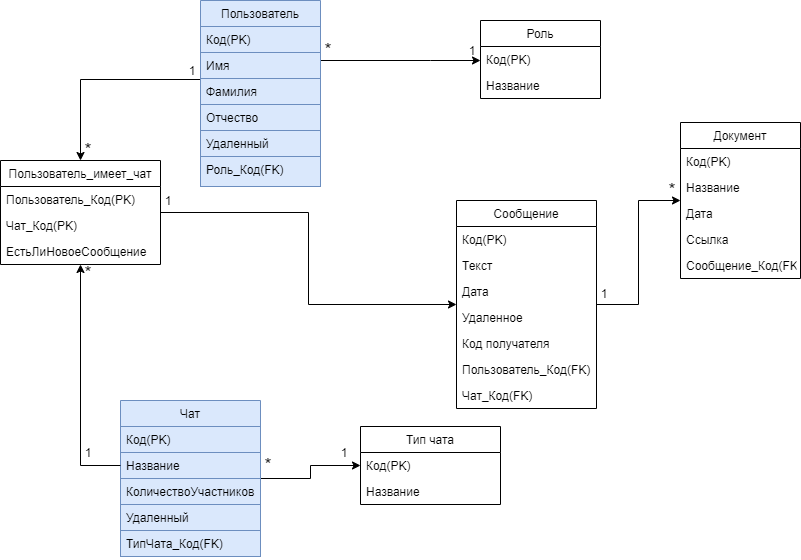


Рисунок 3.3 – Логическая схема базы данных

Таблица «Роль» содержит следующие поля: Код (первичный ключ таблицы), Название.

Таблица «Чат» содержит следующие поля: Код (первичный ключ таблицы), Название, Количество участников, Удаленный, ТипЧата\_Код (внешний ключ таблица «Тип чата»).

Таблица «Пользователь имеет чат» содержит следующие поля: Пользователь\_Код (первичный ключ таблицы, внешний ключ таблицы «Пользователь»), Чат\_Код (первичный ключ таблицы, внешний ключ таблицы «Чат»), Есть ли новое сообщение.

Таблица «Сообщение имеет следующие поля: Код (первичный ключ таблицы), Текст, Дата, Удаленное, Получатель (внешний ключ таблицы «Пользователь»), Пользователь\_Код (внешний код таблицы «Пользователь имеет чат»), Чат) Код (внешний код таблицы «Пользователь имеет чат»).

Таблица «Документ» имеет следующие поля: Код (первичный ключ таблицы), Название, Дата, ссылка, Сообщение\_Код (внешний ключ таблицы «Сообщение)».

Также, в разрабатываемой программной системе будет предусмотрено только программное удаление экземпляров сущностей. Это означает, что при выборе функции удаления какого-либо объекта (например, сообщения) он не будет удаляться полноценно из базы данных, а вместо этого будет у данной записи в базе данных будет выставлен флаг «Удален» в значение true.

В первую очередь, данный функционал позволит избежать непредумышленного удаления записей из базы данных. Помимо этого, это также позволяет контролировать используемые данные (в частности, сообщения).

При необходимости, можно настроить физическое удаление из базы данных по истечении определенного промежутка времени (например, до выпуска группы).

После проектирования логической схемы базы данных, необходимо перевести ее в физическую схему, т.е. то представление, в котором она будет реализовываться с указанием типов полей, первичных и внешних ключей [6].

Физическая схема базы данных представлена на рис. 3.4.

В физической схеме базы данных можно заметить появление в некоторых сущностях новых столбцов, а именно created\_at, updated\_at и deleted\_at. Данные столбцы являются временными метками, которые хранят дополнительную информацию в таблицах, записи которых периодически меняются, добавляются или удаляются (Пользователь, Чат, Пользователь\_Чат, Сообщение, Документ).

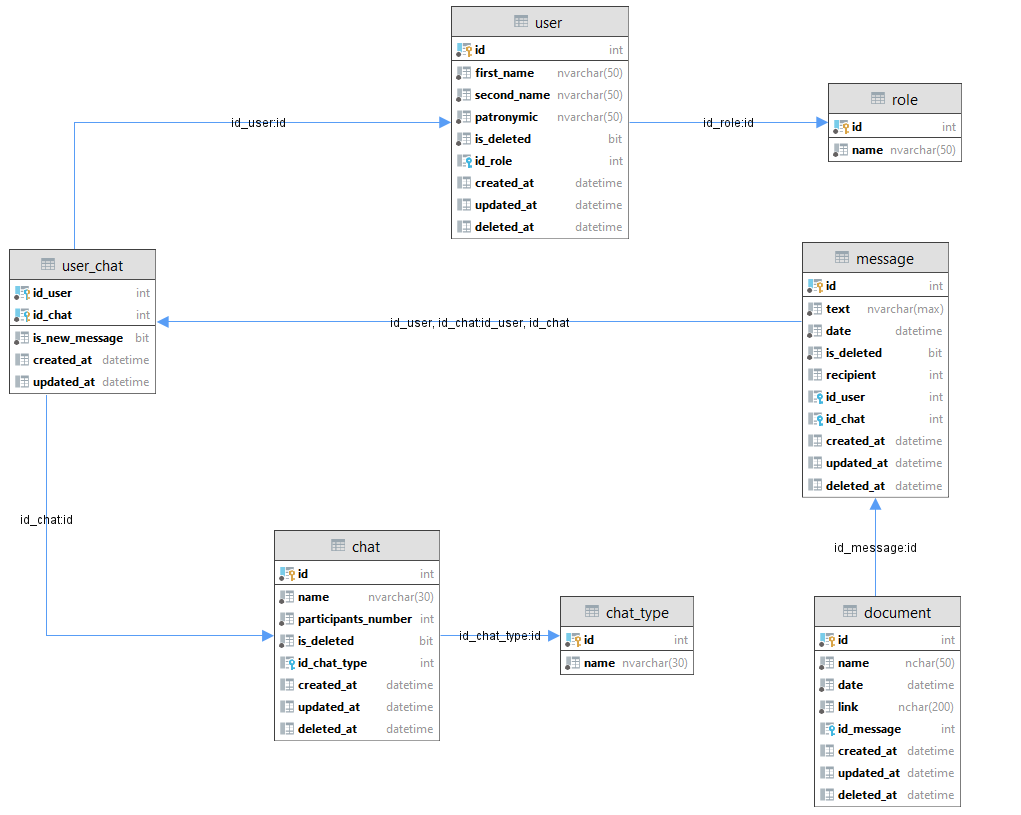


Рисунок 3.4 – Физическая схема базы данных

## Проектирование алгоритмов

Рассмотрим подробнее алгоритм обработки поступившего на сервер запроса. В первую очередь, поскольку обращения всегда происходят от какого-либо конкретного пользователя, то для любого поступившего запроса необходимо произвести проверку на существование такого пользователя, обратившегося на сервер, в базе данных. Если он существует, то мы продолжаем работу, иначе выдаем ошибку.

Далее идем выбор обработчика запроса в зависимости от выбранного маршрута. Если запрос включает в себя код чата, то мы также проверяем существование такого чата у текущего пользователя в базе данных и, исходя из результатов проверки, либо продолжаем работу, либо выдаем ошибку. Аналогичный набор действий производится и при наличии в маршруте кода сообщения и кода документа (данные проверки проводятся с учетом полученных выше данных, т.е. наличие сообщения проводится для конкретного чата и т.д.).

Следующим шагом будет являться проведение контроля соответствия роли пользователя значению «администратор» для таких функций, как создание, изменение, удаление чата, а также добавление, редактирование и удаление пользователя, поскольку только администратор подсистемы управления чатами может выполнять данные действия.

Помимо всего этого, немаловажным будет удостовериться в наличии в теле или строке запроса тех или иных параметров, необходимых для выполнения конкретных действий соответствующих функций. Например, при добавлении чата необходимо проверить наличие названия чата в теле запроса и, если его нет, то мы не можем создать чат и необходимо вернуть ошибку.

Следующим этапом является установление необходимых данных в других зависимых частях системы (например, установление флага наличия новых сообщений в чате таблицы Пользователь\_Чат в значение истины) и формирование тела ответа из запрашиваемого ресурса (ресурсов).

Таким образом, схема вышеописанного алгоритма обработки запроса на сервер сервиса обмена сообщениями представлен на рис. 3.5. Представленные алгоритм составлен для самого подробного случая запроса и, соответственно, некоторые его части могу отсутствовать, а некоторые, наоборот, быть более подробными в зависимости от маршрута.

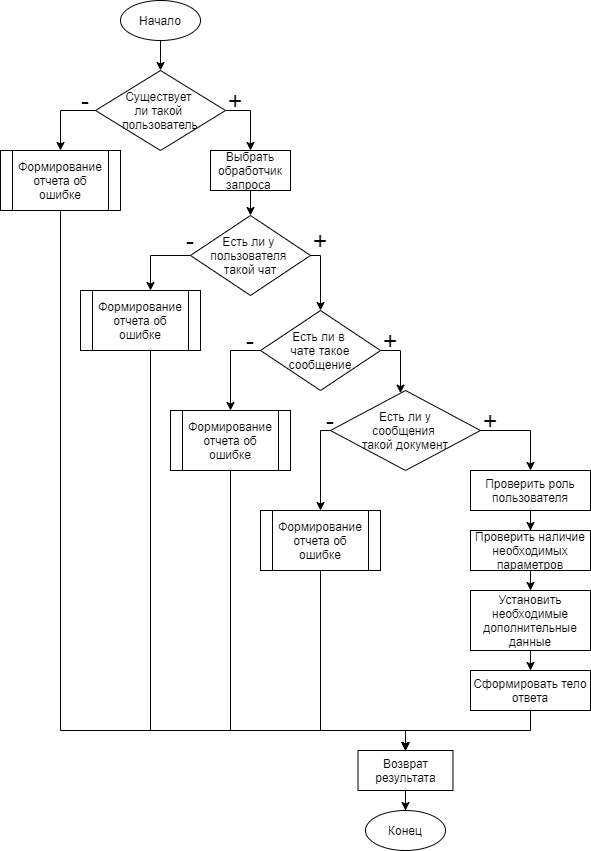


Рисунок 3.5 – Схема алгоритма обработки запроса

# РЕАЛИЗАЦИЯ, ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Реализация основных компонентов серверной подсистемы

Для реализации серверной подсистемы сервера обмена сообщениями были выбраны язык программирования PHP, фреймворк Laravel и интегрированная среда разработки PHPStorm.

Рассмотрим реализацию основных компонентов серверной подсистемы на примере некоторых классов:

* контроллер – ChatController
* модель – Chat.

Полная информация о реализации остальных компонентов программного обеспечения содержится в Приложении 2.

## Класс-контроллер ChatController

Данный класс отвечает за организацию работы управления с чатами и реализует следующие методы:

* index (int userId) – получает код текущего пользователя, запрашивающего ресурсы, формирует список чатов данного пользователя и возвращает их в качестве JSON-массива;
* store (Request request, int userId) – получает код пользователя и массив с информацией о запросе. Берет из данного массива всю необходимую информацию для создания нового экземпляра чата и добавляет его в базу данных. Возвращает информацию о созданной записи;
* show (Request request, int userId, int chatId) – получает код пользователя, код чата и массив с информацией о запросе. Ищет по базе данных чат, соответствующий коду у полученного пользователя, и формирует ответ. Помимо информации о чате из базы данных также добавляет в ответ массив сообщений чата, участников и вложения. Можно передать параметр last, который ограничит количество последний сообщений, добавляемых в массив;
* update (Request request, int userId, int chatId) – получает код пользователя, код чата и массив с информацией о запросе. Ищет по базе данных чат, соответствующий коду у полученного пользователя, и изменяет его информацию на основе данных, полученных из массива request. Сохраняет результат в базу данных и возвращает его в качестве ответа;
* destroy (int userId, int chatId) – получает код пользователя и код чата. Находит по базе данных пользователя и ищет у него чат по переданному коду. Производит программное удаление, т.е. устанавливает в базе данных флаг is\_deleted в значение «истина» и помечает эту запись, как удаленную. При этом она остается в базе данных и, при необходимости, к ней можно получить доступ;
* haveNewMessages (int userId, int chatId) – получает код пользователя и код чата. Находит в базе данных запись по полученным значениям кода и возвращает значение флага is\_new\_message, которое свидетельствует о наличии или отсутствии у данного пользователя в конкретном чате новых сообщений Код данной функции представлен ниже [7].

/\*\*

\* Проверка на наличие новых сообщений в чате пользователя

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function haveNewMessages (int $userId, int $chatId)

{

$flag = UserChat::where ('id\_user', '=', $userId)

->where('id\_chat', '=', $chatId)

->get()[0]['is\_new\_message'];

return response($flag, '200');

}

## Класс-модель Chat

Данный класс обеспечивает связь с базой данных с соответствующей таблицей в ней и имеет следующие поля и методы:

Поля:

* table – хранит имя таблицы в базе данных, с которой связан текущий класс;
* dateFormat – содержит формат хранения столбцов даты-времени таблицы;

Методы:

* chatType () – получает из модели код типа чата и в соответствующей таблице ищет его значение и возвращает в качестве ответа;
* users () – получает из модели код чата и по таблице Пользователь-Чат ищет всех пользователей, которые являются его участниками.

Код данного класса представлен ниже [8]:

/\*\*

\* Название таблицы в базе данных

\* @var string

\* \*/

protected $table = 'chat';

/\*\*

\* Формат хранения столбцов даты модели.

\*

\* @var string

\*/

protected $dateFormat = 'Y-d-m H:m:s';

/\*\*

\* Получить тип чата

\* \*/

public function chatType()

{

return $this->belongsTo(ChatType::class, 'id\_chat\_type');

}

/\*\*

\*Получить всех пользователей чата

\* \*/

public function users()

{

return $this->belongsToMany(User::class, "user\_chat",

"id\_chat", "id\_user")

->using(UserChatPivot::class)

->withPivot('is\_new\_message');

}

## Реализация API

В фреймворке Laravel маршруты АПИ хранятся в отдельном файле, называемом «api.php». Размещение путей именно в этом файле позволяет избежать добавления в начале каждого из них приставки «api». Все маршруты используют один или несколько посредников, которые выполняют некоторые проверки данных перед основной обработкой запроса (например, проверка существования пользователя или чата по полученному коду в базе данных). Поскольку множество путей проходят одинаковые проверки, то они объединяются в группу общего посредника.

Каждый маршрут состоит из строки URI и массива, в котором указываются класс-контроллер и сама функция данного класса, обрабатывающая данный идентификатор ресурса. Также, при необходимости, к маршруту может добавляться свой собственный посредник (как, например, при удалении чата проверяются права пользователя).

Ниже представлен фрагмент кода реализации АПИ в файле «api.php». Полный код можно просмотреть в Приложении А.

Route::middleware(EnsureUserExists::class)->group(function (){

// Запрос на создание чата

Route::post('/v1/{userId}/requestToCreateChat', [OtherFunctionalityController::class, 'requestToAdmin']);

// Чат

Route::get('/v1/{userId}/chats', [ChatController::class, 'index']);

Route::post('/v1/{userId}/chats', [ChatController::class, 'store'])

->middleware('isAdmin');

Route::middleware(EnsureChatExists::class)->group(function (){

// Чат

Route::get('/v1/{userId}/chats/{chatId}', [ChatController::class, 'show']);

Route::put('/v1/{userId}/chats/{chatId}', [ChatController::class, 'update'])

->middleware('isAdmin');

Route::delete('/v1/{userId}/chats/{chatId}', [ChatController::class, 'destroy'])

->middleware('isAdmin');

Route::get('/v1/{userId}/chats/{chatId}/haveNewMessages', [ChatController::class, 'haveNewMessages']);

…

## Тестирование и отладка программного обеспечения

Для тестирования реализованного программного обеспечения воспользуемся сервисом Postman. Postman – это онлайн-сервис, который содержит большой набор инструментов для написания, отладки и тестирования API. Также, он позволяет публиковать его. Данный сервис имеет удобный и понятный графический интерфейс, который значительно упрощает процесс управления созданным API [9].

Рассмотрим возможности Postman и процесс тестирования на примере получения конкретного чата. На рис. 4.1 представлен графический интерфейс применяемой среды разработки API.

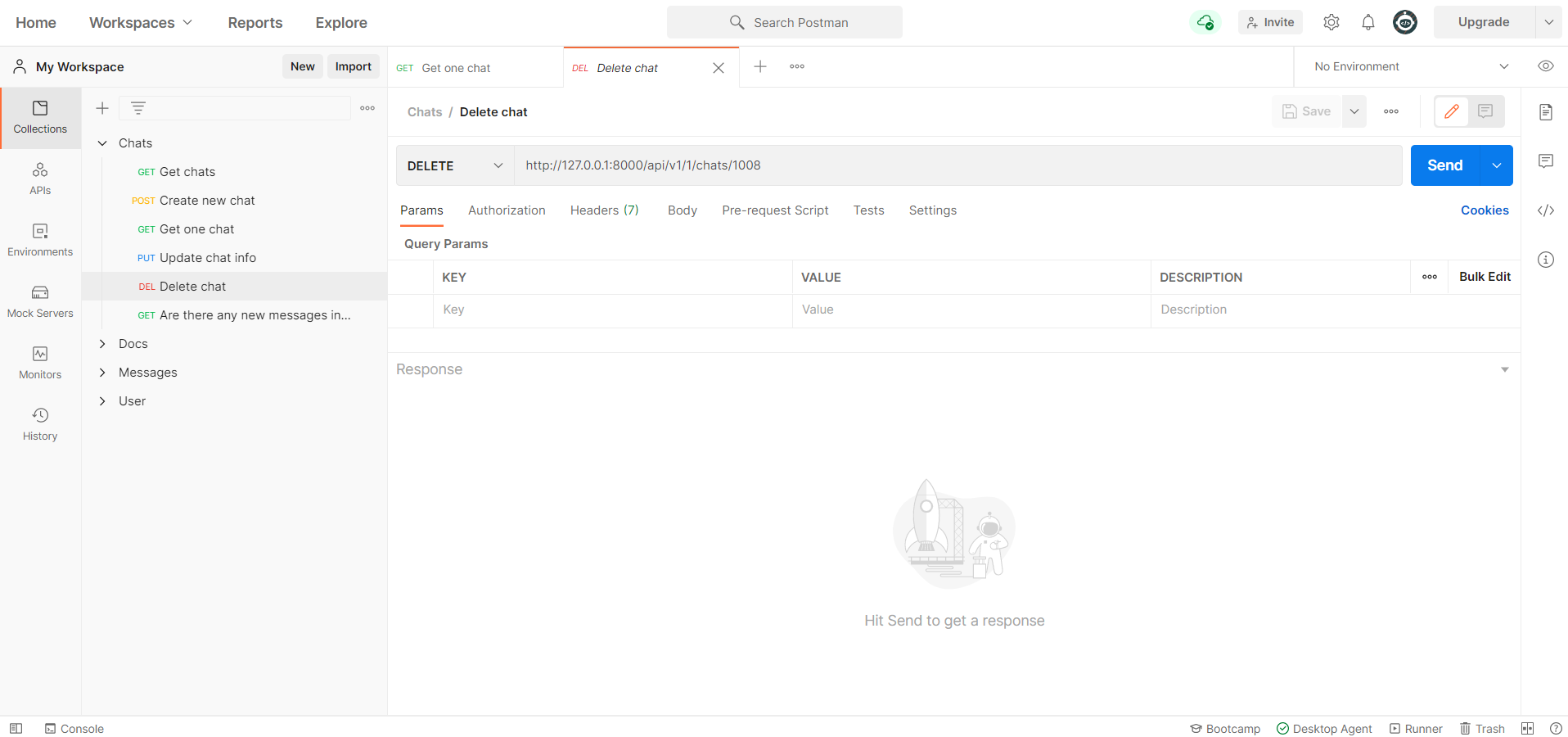


Рисунок 4.1 – Графический интерфейс Postman

Слева представлен список папок, которые объединяют запросы в одну группу по какому-либо критерию (в данном случае, они сгруппированы по основному ресурсу, с которым происходит работа): Chat, Docs, Messages, User. Каждая папка, в свою очередь, содержит список маршрутов, которые описываются названием и методом запроса.

При нажатии на запрос открывается новая вкладка с полной информацией о его структуре. Например, если открыть запрос получения конкретного чата, то можно увидеть следующую его основную информацию (рис. 4.2):

* название в сервисе– Get one chat;
* метод – GET;
* URL – http://127.0.0.1:8000/api/v1/1/chats/1?last=2;
* список параметров запроса – last = 2 (ограничение на количество последних выдаваемых сообщений чата);
* вкладка headers, где описываются пересылаемые серверу необходимые заголовки;
* вкладка body, которая содержит тело запроса со списком передаваемых параметров.

При нажатии на кнопку Send происходит выполнение запроса на сервер по указанному URL (в данном случае, происходит запрос к локальному серверу).

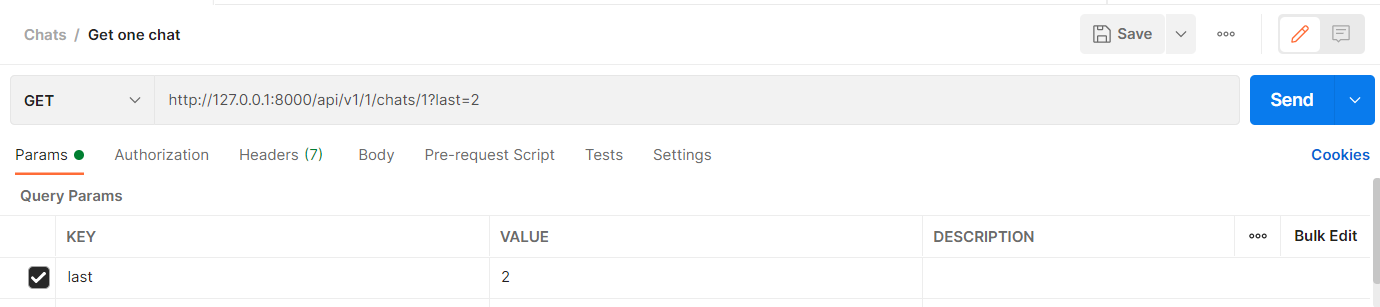


Рисунок 4.2 – Информация о конкретном запросе

На рис. 4.3 представлен полученный ответ в формате JSON от сервера.

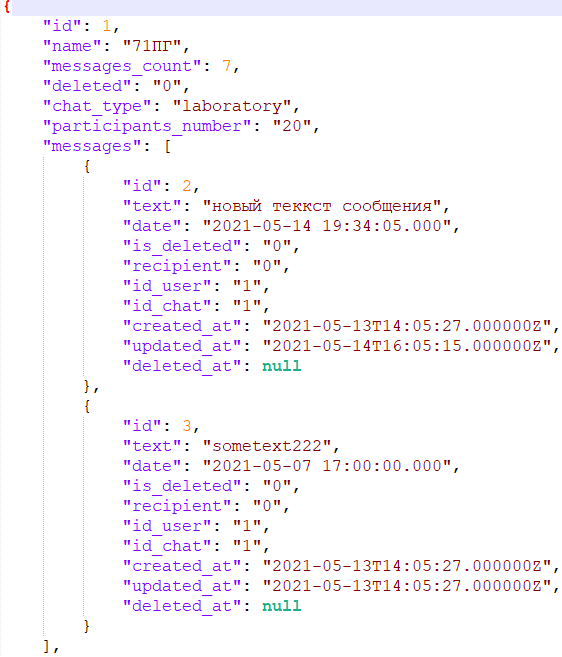
 

Рисунок 4.3 – Ответ сервера

В данном случае мы получили успешный ответ от сервера, т.е. вернулись правильные данные и не произошло никакой ошибки.

Теперь попробуем обратиться к серверу, указав несуществующего пользователя (параметр запроса, стоящий между v1 и chats). Результат показан на рис. 4.4. Как и ожидалось, мы получили ошибку, что такого пользователя не существует. Проведем аналогичный тест для несуществующего чата (параметр после chats) и точно также получим ошибку в ответе (рис. 4.5).

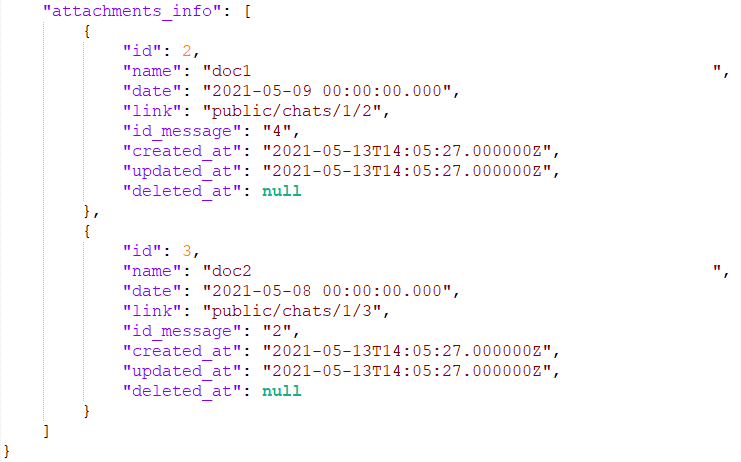


Рисунок 4.3 – Ответ сервера (продолжение)

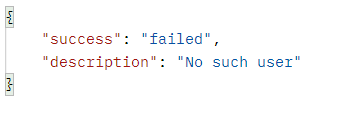


Рисунок 4.4 – Ответ сервера при несуществующем пользователе

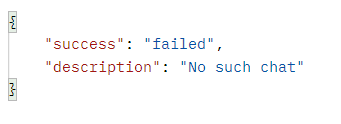


Рисунок 4.5 – Ответ сервера при несуществующем чате

Атрибуты качества программного обеспечения, заявленные в цели данной работы, а именно:

* удобство внесения изменений (changeability);
* удобство проверки (testability).

достигаются засчет использования клиент-серверной архитектуры и разделения всего сервиса обмена сообщениями на отдельные компоненты. Это означает, что из-за того факта, что серверная подсистема никаким образом не зависит от клиента, а также благодаря наличию разделения программных компонентов на модули (контроллеры, модели, API) появляется возможность значительного снижения трудозатрат на внесение дальнейших изменений в программную систему (добавление новых функций, изменение старых и т.д.), а также на организацию процесса проверки (или тестирования) уже имеющихся или добавленных функциональных возможностей.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были решены следующие задачи:

1. проанализирована задача разработки специализированного сервиса обмена сообщениями;
2. изучены функциональные возможности существующего решения;
3. сформулированы требования и определены функциональные и нефункциональные возможности разрабатываемой серверной подсистемы;
4. разработаны архитектура серверной подсистемы и базы данных сервиса обмена сообщениями;
5. разработаны функциональные и информационные спецификации;
6. спроектированы собственное API сервера и основные алгоритмы;
7. реализован программный продукт.

Таким образом, поставленные задачи выполнены, а цель работы достигнута.

# СПИСОК ЛИТЕРАТУРЫ

1. Архитектура WhatsApp, которую Facebook купил за $19 миллиардов [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/276951/. – Дата доступа: 11.05.2021.
2. Интервью с CTO и сооснователем Viber Игорем Магазинником [Электронный ресурс]. – Режим доступа: https://xakep.ru/2014/06/26/intervyu-viber/. – Дата доступа: 11.05.2021.
3. Клиент-сервер – Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент\_—\_сервер. – Дата доступа: 11.05.2021.
4. Разработка REST API — что такое Code First подход? [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/483322/. – Дата доступа: 11.05.2021.
5. Разработка REST API — что такое Contract First? [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/483206/. – Дата доступа: 11.05.2021.
6. Техническая документация по SQL Server [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver15. – Дата доступа: 11.05.2021.
7. PHP: Руководство по PHP – Manual [Электронный ресурс]. – Режим доступа: https://www.php.net/manual/ru/index.php. – Дата доступа: 11.05.2021.
8. Официальная документация Laravel [Электронный ресурс]. – Режим доступа: https://laravel.com/docs/8.x. – Дата доступа: 11.05.2021.
9. Documenting your API | Postman Learning Center [Электронный ресурс]. – Режим доступа: https://learning.postman.com/docs/publishing-your-api/documenting-your-api/. – Дата доступа: 11.05.2021.

# Приложение 1

**(обязательное)**

# Документация API

openapi: 3.0.0

info:

version: '1.0'

title: 'Messenger`s server API'

description: API for server of OSU messenger

##эндпоинты для сообщений

servers:

# Added by API Auto Mocking Plugin

- description: SwaggerHub API Auto Mocking

url: https://virtserver.swaggerhub.com/MyPersonalOrg3/MessengerServer/1.0

tags:

- name: chat

description: Everything about chat

- name: user

description: Everything about user

- name: document

description: Everything about document

- name: message

description: Everything about message

- name: adminFunctionality

description: Functions that allowed only for admin

paths:

/api/v1/{userId}/requestToCreateChat:

parameters:

- $ref: '#/components/parameters/userId'

post:

summary: Send request to admin to create new personal chat with some user

responses:

'201':

description: Successfully created chat

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/ChatInfo'

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/User'

########## Все чаты пользователя ##########

/api/v1/{userId}/chats:

parameters:

- $ref: '#/components/parameters/userId'

get:

tags:

- chat

summary: Returns a list of chatsInfo of user

responses:

'200':

description: Successfully returned a list of chats of this student

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/ChatInfo'

post:

tags:

- chat

- adminFunctionality

summary: Add new chat

responses:

'201':

description: Created chat and returns some metainfo

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/NewChat'

########## Конкретный чат ##########

/api/v1/{userId}/chats/{chatId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

- name: last

in: query

description: Count of last messages in chat which should be returned

schema:

type: integer

minimum: 1

example: 1

get:

tags:

- chat

summary: Returns a chat by {chatId}

responses:

'200':

description: Successfully returned chat

content:

application/json:

schema:

$ref: '#/components/schemas/Chat'

put:

tags:

- chat

- adminFunctionality

summary: Change information about chat by {chatId}

responses:

'200':

description: Successfully changed info about chat

'403':

description: Allowed only for admin

'404':

description: There is no such chat

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/ChatInfo'

delete:

tags:

- chat

- adminFunctionality

summary: Delete chat

description: This only can be done by admin

responses:

'200':

description: Chat deleted successfully

'403':

description: Allowed only for admin

'404':

description: There is no such chat

######### Проверка на наличие новых сообщений

/api/v1/{userId}/chats/{chatId}/haveNewMessages:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- chat

summary: Returns a flag for new messages for chat

responses:

'200':

description: Successfully returned flag

content:

application/json:

schema:

type: boolean

########## Работа со списком сообщений ##########

/api/v1/{userId}/chats/{chatId}/messages:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

- name: last

in: query

description: Count of last messages in chat which should be returned

schema:

type: integer

minimum: 1

example: 1

get:

tags:

- message

summary: Get list of messages and set flag "is new messages" to false

responses:

'200':

description: Successfully returned list of messages in current chat

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/Message'

post:

tags:

- message

summary: Add new message to chat and set flag "is new messages" to true

responses:

'200':

description: Successfully added new message to current chat

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/NewMessage'

########## Работа с конкретным сообщением ##########

/api/v1/{userId}/chats/{chatId}/messages/{messageId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

- $ref: '#/components/parameters/messageId'

get:

tags:

- message

summary: Get message

responses:

'200':

description: Successfully returned message in current chat

content:

application/json:

schema:

$ref: '#/components/schemas/Message'

'404':

description: There is no such message

put:

tags:

- message

summary: Change text in message

responses:

'200':

description: Successfully changed message`s text

'404':

description: There is no such message

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/Message'

delete:

tags:

- message

summary: Delete message

responses:

'200':

description: Successfully deleted message

'404':

description: There is no such message

########## Пользователи одного чата ##########

/api/v1/{userId}/chats/{chatId}/users:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- user

summary: Returns all users of {chatId} chat

responses:

'200':

description: Successfully returned users

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/User'

post:

tags:

- user

- adminFunctionality

summary: Add new user to {chatId} chat

responses:

'200':

description: Successfully added new user

#код ошибки добавления

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/NewUser'

########## Конкретный пользователь чата ##########

/api/v1/{userId}/chats/{chatId}/users/{userInChatId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

- $ref: '#/components/parameters/userInChatId'

get:

tags:

- user

summary: Returns info about {userId} user

responses:

'200':

description: Successfully returned info users

content:

application/json:

schema:

$ref: '#/components/schemas/User'

put:

tags:

- user

- adminFunctionality

summary: Change info about {userId} user

responses:

'200':

description: Successfully changed info

'404':

description: There is no such user

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/NewUser'

delete:

tags:

- user

- adminFunctionality

summary: Delete {userId} user

responses:

'200':

description: Successfully deleted user

'403':

description: Allowed only for admin

'404':

description: There is no such user

########## Документы чата ##########

/api/v1/{userId}/chats/{chatId}/documents:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- document

summary: Returns all documents info of chat {chatId}

responses:

'200':

description: Successfully returned documents

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/DocumentMetainfo'

post:

tags:

- document

summary: Create metadata about new document in chat {chatId}

responses:

'201':

description: Created metadata and returns that with doc`s id

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/NewDocumentMetainfo'

########## Метаданные об одном документа чата ##########

/api/v1/{userId}/chats/{chatId}/documents/{documentId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/documentId'

- $ref: '#/components/parameters/userId'

get:

tags:

- document

description: Returns a metadata about {documentId} document

responses:

'200':

description: Successfully returned a metadata about {documentId} document

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/DocumentMetainfo'

put:

tags:

- document

description: Change a metadata about {documentId} document

responses:

'200':

description: Successfully changed a metadata about {documentId} document

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

delete:

tags:

- document

description: Delete {documentId} document

responses:

'200':

description: Successfully deleted {documentId} document

########## Один документ ##########

/api/v1/{userId}/chats/{chatId}/documents/{documentId}/data:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/documentId'

- $ref: '#/components/parameters/userId'

get:

tags:

- document

description: Returns a {documentId} document

responses:

'200':

description: Successfully returned a document by {documentId} document

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

post:

tags:

- document

description: Load a {documentId} document to server

responses:

'200':

description: Successfully loaded the document

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

########## Все метаданные документов пользователя в чате ##########

/api/v1/{userId}/chats/{chatId}/userDocuments:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- chat

- user, document

summary: Info about all user's documents in chat

description: Returns info about all documents of {userId} user

responses:

'200':

description: Successfully returned info of all docs

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/DocumentMetainfo'

components:

schemas:

Chat:

type: object

properties:

id:

type: integer

minimum: 1

example: 1

name:

type: string

example: 71ПГ

messages:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/Message'

# ---- /Added line --------------------------------

messages\_count:

type: integer

minimum: 0

example: 10

deleted:

type: boolean

example: false

chat\_type:

type: string

enum: [laboratory, practice, lecture, personal, exam, consultation]

example: practic

participants:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/User'

# ---- /Added line --------------------------------

attachments\_info:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/DocumentMetainfo'

# ---- /Added line --------------------------------

ChatInfo:

type: object

properties:

id:

type: integer

minimum: 1

example: 1

name:

type: string

example: 71ПГ

messages\_count:

type: integer

minimum: 0

example: 10

deleted:

type: boolean

example: false

chat\_type:

type: string

enum: [laboratory, practice, lecture, personal, exam, consultation]

example: practic

participants\_number:

type: integer

minimum: 0

example: 10

NewChat:

type: object

properties:

name:

type: string

example: 71ПГ

chat\_type:

type: string

enum: [laboratory, practice, lecture, personal, exam, consultation]

example: practic

participants\_number:

type: integer

minimum: 0

example: 10

Message:

type: object

properties:

id:

type: integer

minimum: 1

example: 1

chat\_id:

type: integer

minimum: 1

example: 1

user\_recipient\_id:

type: integer

minimum: 1

example: 1

deleted:

type: boolean

example: false

data:

type: string

example: Какой-то текст

date:

type: string

format: date-time

example: 2021-05-14 17:21:20 #Y-m-d H:m:s

attachments\_info:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/DocumentMetainfo'

# ---- /Added line --------------------------------

NewMessage:

type: object

properties:

chat\_id:

type: integer

minimum: 1

example: 1

user\_recipient\_id: #0 - Сообщение для всего чата, какое-то значение - получатель сообщения

type: integer

minimum: 0

example: 1

data:

type: string

example: Какой-то текст

date:

type: string

format: date-time

example: 2021-05-14 17:21:20 #Y-m-d H:m:s

UpdateMessage:

type: object

properties:

data:

type: string

example: Какой-то текст

date:

type: string

format: date-time

example: 2021-05-14 17:21:20 #Y-m-d H:m:s

User:

type: object

properties:

id:

type: integer

example: 1

minimum: 1

role:

type: string

enum: [student, teacher, admin]

example: student

name:

type: string

example: Владислав

surname:

type: string

example: Шорин

patronymic:

type: string

example: Дмитриевич

deleted:

type: boolean

example: false

NewUser:

type: object

properties:

role:

type: string

enum: [student, teacher, admin]

example: student

name:

type: string

example: Владислав

surname:

type: string

example: Шорин

patronymic:

type: string

example: Дмитриевич

DocumentMetainfo:

type: object

properties:

id:

type: integer

example: 1

minimum: 1

name:

type: string

date:

type: string

example: 2021-05-14 17:21:20 #Y-m-d H:m:s

link:

type: string

message\_id:

type: integer

NewDocumentMetainfo:

type: object

properties:

name:

type: string

date:

type: string

example: 2021-05-14 17:21:20 #Y-m-d H:m:s

message\_id:

type: integer

parameters:

chatId:

name: chatId

in: path

required: true

description: Numeric ID of the chat to get

schema:

type: integer

minimum: 1

documentId:

name: documentId

in: path

required: true

description: Numeric ID of the document to get

schema:

type: integer

minimum: 1

userId:

name: userId

in: path

required: true

description: Numeric ID of the user to get

schema:

type: integer

minimum: 1

userInChatId:

name: userInChatId

in: path

required: true

description: Numeric ID of the user in chat to get

schema:

type: integer

minimum: 1

messageId:

name: messageId

in: path

required: true

description: Numeric ID of the message to get

schema:

type: integer

minimum: 1

# Приложение 2

**(обязательное)**

# Исходный текст программы

**«api.php»**

<?php

use App\Http\Controllers\ChatController;

use App\Http\Controllers\DocumentController;

use App\Http\Controllers\MessageController;

use App\Http\Controllers\OtherFunctionalityController;

use App\Http\Controllers\UserController;

use App\Http\Middleware\EnsureChatExists;

use App\Http\Middleware\EnsureDocumentExists;

use App\Http\Middleware\EnsureUserExists;

use Illuminate\Support\Facades\Route;

Route::middleware(EnsureUserExists::class)->group(function (){

// Запрос на создание чата

Route::post('/v1/{userId}/requestToCreateChat', [OtherFunctionalityController::class, 'requestToAdmin']);

// Чат

Route::get('/v1/{userId}/chats', [ChatController::class, 'index']);

Route::post('/v1/{userId}/chats', [ChatController::class, 'store'])

->middleware('isAdmin');

Route::middleware(EnsureChatExists::class)->group(function (){

// Чат

Route::get('/v1/{userId}/chats/{chatId}', [ChatController::class, 'show']);

Route::put('/v1/{userId}/chats/{chatId}', [ChatController::class, 'update'])

->middleware('isAdmin');

Route::delete('/v1/{userId}/chats/{chatId}', [ChatController::class, 'destroy'])

->middleware('isAdmin');

Route::get('/v1/{userId}/chats/{chatId}/haveNewMessages', [ChatController::class, 'haveNewMessages']);

// Сообщения

Route::get('/v1/{userId}/chats/{chatId}/messages', [MessageController::class, 'index']);

Route::post('/v1/{userId}/chats/{chatId}/messages', [MessageController::class, 'store']);

Route::get('/v1/{userId}/chats/{chatId}/messages/{messageId}', [MessageController::class, 'show'])

->middleware('messageExists');

Route::put('/v1/{userId}/chats/{chatId}/messages/{messageId}', [MessageController::class, 'update'])

->middleware('messageExists');

Route::delete('/v1/{userId}/chats/{chatId}/messages/{messageId}', [MessageController::class, 'destroy'])

->middleware('messageExists');

// Пользователь

Route::get('v1/{userId}/chats/{chatId}/users', [UserController::class, 'index']);

Route::post('v1/{userId}/chats/{chatId}/users', [UserController::class, 'store'])

->middleware('isAdmin');

Route::get('v1/{userId}/chats/{chatId}/users/{userInChatId}', [UserController::class, 'show']);

Route::put('v1/{userId}/chats/{chatId}/users/{userInChatId}', [UserController::class, 'update'])

->middleware('isAdmin');

Route::delete('v1/{userId}/chats/{chatId}/users/{userInChatId}', [UserController::class, 'destroy'])

->middleware('isAdmin');

// Документы

Route::get('v1/{userId}/chats/{chatId}/documents', [DocumentController::class, 'index']);

Route::post('v1/{userId}/chats/{chatId}/documents', [DocumentController::class, 'store']);

Route::get('v1/{userId}/chats/{chatId}/userDocuments', [DocumentController::class, 'userDocuments']);

Route::middleware(EnsureDocumentExists::class)->group(function (){

Route::get('v1/{userId}/chats/{chatId}/documents/{documentId}', [DocumentController::class, 'show']);

Route::put('v1/{userId}/chats/{chatId}/documents/{documentId}', [DocumentController::class, 'update']);

Route::delete('v1/{userId}/chats/{chatId}/documents/{documentId}', [DocumentController::class, 'destroy']);

Route::get('v1/{userId}/chats/{chatId}/documents/{documentId}/data', [DocumentController::class, 'getData']);

Route::post('v1/{userId}/chats/{chatId}/documents/{documentId}/data', [DocumentController::class, 'setData']);

});

});

});

**«ChatController.php»**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Chat;

use App\Models\ChatType;

use App\Models\Document;

use App\Models\Message;

use App\Models\User;

use App\Models\UserChat;

use App\Http\Resources\ChatCollection;

use App\Http\Resources\ChatResource;

class ChatController extends Controller{

/\*\*

\* Display a listing of the resource.

\*

\* @param int $userId

\* @return \Illuminate\Http\Response

\*/

public function index(int $userId) {

$user = User::find($userId);

$chats = $user->chats;

$data = (new ChatCollection($chats))

->additional(['meta' => [

'asd' => 'asd',

]]);

return response($data, '200');

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request, int $userId) {

$chat\_type = ChatType::where('name', '=', $request->chat\_type)->get();

$chat = new Chat();

$chat->name = $request->name;

$chat->participants\_number = $request->participants\_number;

$chat->id\_chat\_type = $chat\_type[0]['id'];

$chat->is\_deleted = false;

$chat->save();

$data = Chat::find(Chat::max('id'));

return response($data, '201');

}

/\*\*

\* Display the specified resource.

\*

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function show(Request $request, int $userid, int $chatId) {

$chat = new ChatResource(Chat::find($chatId));

$lastMessagesCount = $request->input('last');

$messages = Message::where('id\_user', '=', $userid)

->where('id\_chat', '=', $chatId)

->when($lastMessagesCount, function ($query, $last){

return $query->take($last);

})

->get();

$chatUsers = UserChat::where('id\_chat', '=', $chatId)->get();

$participants = array();

foreach ($chatUsers as $user) {

$participant = User::find($user['id\_user']);

array\_push($participants, $participant);

}

$attachments = array();

foreach ($messages as $message) {

$document = Document::find($message['id']);

if (empty($document)) continue;

array\_push($attachments, $document);

}

$chat->additional([

'messages' => $messages,

'participants' => $participants,

'attachments\_info' => $attachments,

]);

$data = array\_merge($chat->toArray($chat), $chat->additional);

return response($data, '200');

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, int $userId, int $chatId) {

$chat = Chat::find($chatId);

$chat->name = $request->name;

$id\_chat\_type = ChatType::where('name', '=', $request->chat\_type)->get()[0]['id'];

$chat->id\_chat\_type = $id\_chat\_type;

$chat->save();

return response($chat, '200');

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function destroy(int $userId, int $chatId) {

$chat = Chat::find($chatId);

$chat->is\_deleted = true;

$chat->save();

$chat->delete();

// $messagesToDelete = Message::where('id\_chat', '=', $chatId)

// ->get();

return response([

'success' => 'success',

'description' => 'Successfully deleted chat'

], '200');

}

/\*\*

\* Проверка на наличие новых сообщений в чате пользователя

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/ public function haveNewMessages(int $userId, int $chatId)

{

$flag = UserChat::where('id\_user', '=', $userId)

->where('id\_chat', '=', $chatId)

->get()[0]['is\_new\_message'];

return response($flag, '200');

}

}

**«DocumentController.php»**

<?php

namespace App\Http\Controllers;

use App\Models\User;

use Illuminate\Http\Request;

use App\Models\Document;

use App\Models\Message;

use Illuminate\Support\Facades\Storage;

use PhpParser\Comment\Doc;

class DocumentController extends Controller

{

/\*\*

\* Display a listing of the resource.

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function index(int $userId, int $chatId) {

$messages = Message::where('id\_chat', '=', $chatId)->get();

$docs = [];

foreach ($messages as $message) {

if (empty($message->attachmentsInfo[0])) continue;

array\_push($docs, $message->attachmentsInfo);

}

$data = $docs;

return response($data, '200');

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request, int $userId, int $chatId) {

$newDocument = new Document();

$newDocument->name = $request->name;

$newDocument->date = $request->date;

$newDocument->id\_message = $request->message\_id;

$newDocument->link = "";

$newDocument->save();

$doc = Document::find(Document::max('id'));

return response($doc,'201');

}

/\*\*

\* Display the specified resource.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $documentId

\* @return \Illuminate\Http\Response

\*/

public function show(int $userId, int $chatId, int $documentId) {

$document = Document::find($documentId);

return response($document, '200');

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @param int $documentId

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, int $userId, int $chatId, int $documentId) {

$document = Document::find($documentId);

$document->name = $request->name;

$document->date = $request->date;

$document->save();

return response($document, '200');

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $documentId

\* @return \Illuminate\Http\Response

\*/

public function destroy(int $userId, int $chatId, int $documentId) {

$doc = Document::find($documentId);

$doc->delete();

return response([

'success' => 'success',

'description' => 'Successfully deleted document'

], '200');

}

/\*\*

\* Get document by id

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @param int $documentId

\* @return \Illuminate\Http\Response

\*/

public function getData(Request $request, int $userId, int $chatId, int $documentId) {

$doc = Document::find($documentId);

return Storage::download($doc->link, $doc->name);

// return Storage::download($doc->link, $doc->name.$doc->extendion);

}

/\*\*

\* Add document

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @param int $documentId

\* @return \Illuminate\Http\Response

\*/

public function setData(Request $request, int $userId, int $chatId, int $documentId) {

$link = '';

$doc = Document::find($documentId);

if ($request->hasFile('file')) {

$file = $request->file('file');

$name = $file->getClientOriginalName();

$link = $file->storeAs($chatId.'/'.$userId, $name);

$doc->link = $link;

$doc->name = $name;

$doc->extension = $file->extension();

$doc->save();

return response($doc, '200');

}

else {

return response([

'success' => 'failed',

'description' => 'There is no file'

], '404');

}

}

/\*\*

\* Get user`s docs in chat

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function userDocuments(int $userId, int $chatId) {

$messages = Message::where('id\_user', '=', $userId)

->where('id\_chat', '=', $chatId)

->get();

$docs = [];

foreach ($messages as $message) { array\_push($docs, $message->attachmentsInfo); }

return response($docs, '200');

}

}

**«MessageController.php»**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Message;

use App\Models\UserChat;

use App\Http\Resources\MessageCollection;

class MessageController extends Controller

{

/\*\*

\* Display a listing of the resource.

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function index(Request $request, int $userId, int $chatId) {

$lastMessagesCount = $request->input('last');

$messages = new MessageCollection(Message::where('id\_user', '=', $userId)

->where('id\_chat', '=', $chatId)

->when($lastMessagesCount, function ($query, $last){

return $query->take($last);

})

->get()

);

UserChat::where('id\_chat', '=', $chatId)

->update(['is\_new\_message' => 0]);

$data = $messages;

return response($data, '200');

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request, int $userId, int $chatId) {

$newMessage = new Message();

$newMessage->text = $request->data;

$newMessage->date = $request->date;

$newMessage->id\_chat = $chatId;

$newMessage->id\_user = $userId;

$newMessage->recipient = $request->user\_recipient\_id;

$newMessage->is\_deleted = 0;

$newMessage->save();

UserChat::where('id\_chat', '=', $chatId)

->update(['is\_new\_message' => 1]);

$data = Message::find(Message::max('id'));

return response($data, '201');

}

/\*\*

\* Display the specified resource.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $messageId

\* @return \Illuminate\Http\Response

\*/

public function show(int $userId, int $chatId, int $messageId) {

$message = Message::find($messageId);

$message->attachmentsInfo->toArray();

return response($message, '200');

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @param int $messageId

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, int $userId, int $chatId, int $messageId) {

$message = Message::find($messageId);

$message->text = $request->data;

$message->date = $request->date;

$message->save();

return response($message, '200');

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $messageId

\* @return \Illuminate\Http\Response

\*/

public function destroy(int $userId, int $chatId, int $messageId) {

$message = Message::find($messageId);

$message->is\_deleted = true;

$message->save();

$message->delete();

return response([

'success' => 'success',

'description' => 'Successfully deleted message'

], '200');

}

}

**«OtherFunctionalityController.php»**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class OtherFunctionalityController extends Controller

{

/\*\*

\* Send request to admin to create new chat

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function requestToAdmin(Request $request) {

return response([

'success' => 'failed',

'description' => 'Sorry, but this function doesnt exists at the moment'

], '404');

}

}

**«UserController.php»**

<?php

namespace App\Http\Controllers;

use App\Http\Resources\UserResource;

use App\Models\Chat;

use App\Models\Role;

use App\Models\User;

use App\Models\UserChat;

use Illuminate\Http\Request;

class UserController extends Controller{

/\*\*

\* Display a listing of the resource.

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function index(int $userId, int $chatId) {

$users = Chat::find($chatId)->users;

$data = UserResource::collection($users);

return response($data, '200');

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request, int $userId, int $chatId) {

$user = new User();

$user->first\_name = $request->name;

$user->second\_name = $request->surname;

$user->patronymic = $request->patronymic;

$user->is\_deleted = false;

$user->id\_role = Role::where('name', '=', $request->role)->get()[0]['id'];

$user->save();

$newUser = User::find(User::max('id'));

$userChat = new UserChat();

$userChat->id\_user = $newUser->id;

$userChat->id\_chat = $chatId;

$userChat->is\_new\_message = false;

$userChat->save();

return response([

$newUser

], '200');

}

/\*\*

\* Display the specified resource.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $userInChatId

\* @return \Illuminate\Http\Response

\*/

public function show(int $userId, int $chatId, int $userInChatId) {

$user = new UserResource(User::find($userInChatId));

return response($user, '200');

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $userId

\* @param int $chatId

\* @param int $userInChatId

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, int $userId, int $chatId, int $userInChatId) {

$user = User::find($userInChatId);

$user->first\_name = $request->name;

$user->second\_name = $request->surname;

$user->patronymic = $request->patronymic;

$user->id\_role = Role::where('name', '=', $request->role)->get()[0]['id'];

$user->save();

return response($user, '200');

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $userId

\* @param int $chatId

\* @param int $userInChatId

\* @return \Illuminate\Http\Response

\*/

public function destroy(int $userId, int $chatId, int $userInChatId) {

$user = User::find($userInChatId);

$user->is\_deleted = true;

$user->save();

$user->delete();

return response([

'success' => 'success',

'description' => 'Successfully deleted user'

], '200');

}

}

**«EnsureChatExists.php»**

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

class EnsureChatExists{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next) {

$chatId = $request->route('chatId');

$chat = \App\Models\Chat::find($chatId);

if (empty($chat)) {

return response([

'success' => 'failed',

'description' => 'No such chat',

],

'404');

}

return $next($request);

}

}

**«EnsureDocumentExists.php»**

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

class EnsureDocumentExists{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next) {

$documentId = $request->route('documentId');

$document = \App\Models\Document::find($documentId);

if (empty($document)) {

return response([

'success' => 'failed',

'description' => 'No such document',

],

'404');

}

return $next($request);

}

}

**«EnsureMessageExists.php»**

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

class EnsureMessageExists{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next) {

$messageId = $request->route('messageId');

$message = \App\Models\Message::find($messageId);

if (empty($message)) {

return response([

'success' => 'failed',

'description' => 'No such message',

],

'404');

}

return $next($request);

}

}

**«EnsureUserExists.php»**

<?php

namespace App\Http\Middleware;

use Closure;

use http\Client\Curl\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\App;

class EnsureUserExists{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next) {

$userId = $request->route('userId');

$user = \App\Models\User::find($userId);

if (empty($user)) {

return response([

'success' => 'failed',

'description' => 'No such user',

],

'404');

}

return $next($request);

}

}

**«EnsureUserIsAdmin.php»**

<?php

namespace App\Http\Middleware;

use App\Models\Role;

use App\Models\User;

use Closure;

use Illuminate\Http\Request;

class EnsureUserIsAdmin{

/\*\*

\* Handle an incoming request.

\*

\* @param Request $request

\* @param Closure $next

\* @return mixed

\*/

public function handle(Request $request, Closure $next) {

$userId = $request->route('userId');

$userRole = Role::find(User::find($userId)->id\_role)->name;

if ($userRole != 'admin') {

return response([

'success' => 'failed',

'description' => 'Allowed only for admin'

],

'403');

}

return $next($request);

}

}

**«Kernel.php»**

<?php

namespace App\Http;

use App\Http\Middleware\EnsureChatExists;

use App\Http\Middleware\EnsureDocumentExists;

use App\Http\Middleware\EnsureMessageExists;

use App\Http\Middleware\EnsureUserExists;

use App\Http\Middleware\EnsureUserIsAdmin;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel{

/\*\*

\* The application's global HTTP middleware stack.

\*

\* These middleware are run during every request to your application.

\*

\* @var array

\*/

protected $middleware = [

// \App\Http\Middleware\TrustHosts::class,

\App\Http\Middleware\TrustProxies::class,

\Fruitcake\Cors\HandleCors::class,

\App\Http\Middleware\PreventRequestsDuringMaintenance::class,

\Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,

\App\Http\Middleware\TrimStrings::class,

\Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,

];

/\*\*

\* The application's route middleware groups.

\*

\* @var array

\*/

protected $middlewareGroups = [

'web' => [

\App\Http\Middleware\EncryptCookies::class,

\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

\Illuminate\Session\Middleware\StartSession::class,

// \Illuminate\Session\Middleware\AuthenticateSession::class,

\Illuminate\View\Middleware\ShareErrorsFromSession::class,

\App\Http\Middleware\VerifyCsrfToken::class,

\Illuminate\Routing\Middleware\SubstituteBindings::class,

],

'api' => [

'throttle:api',

\Illuminate\Routing\Middleware\SubstituteBindings::class,

],

];

/\*\*

\* The application's route middleware.

\*

\* These middleware may be assigned to groups or used individually.

\*

\* @var array

\*/

protected $routeMiddleware = [

'auth' => \App\Http\Middleware\Authenticate::class,

'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,

'can' => \Illuminate\Auth\Middleware\Authorize::class,

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,

'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,

'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

'isAdmin' => EnsureUserIsAdmin::class,

'userExists' => EnsureUserExists::class,

'chatExists' => EnsureChatExists::class,

'messageExists' => EnsureMessageExists::class,

'documentExists' => EnsureDocumentExists::class,

];

}

**«Chat.php»**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\SoftDeletes;

class Chat extends Model{

use HasFactory;

use SoftDeletes;

/\*\*

\* Название таблицы в БД

\* @var string

\* \*/

protected $table = 'chat';

/\*\*

\* Формат хранения столбцов даты модели.

\*

\* @var string

\*/

protected $dateFormat = 'Y-d-m H:m:s';

/\*\*

\* Получить тип чата

\* \*/

public function chatType() { return $this->belongsTo(ChatType::class, 'id\_chat\_type'); }

/\*\*

\*Получить всех пользователей чата

\* \*/

public function users() {

return $this->belongsToMany(User::class, "user\_chat",

"id\_chat", "id\_user")

->using(UserChatPivot::class)

->withPivot('is\_new\_message');

}

}

**«ChatType.php»**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class ChatType extends Model{

use HasFactory;

/\*

\* Название таблицы в БД

\* @var string

\* \*/

protected $table ='chat\_type';

/\*

\* Получить чаты определенного типа

\* \*/

public function chats() { return $this->hasMany(Chat::class, 'id\_chat\_type'); }

}

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**

**«.php»**