МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики

Кафедра программного обеспечения и администрирования информационных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Эксплуатация информационных систем и баз данных»

на тему: ИНФОРМАЦИОННАЯ СИСТЕМА ОБМЕНА ФАЙЛАМИ И СООБЩЕНИЯ ДЛЯ ОРГАНИЗАЦИИ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА

Обучающегося 4 курса очной формы обучения направления подготовки
02.03.03 Математическое обеспечение и администрирование информационных систем
Направленность (профиль)
Проектирование информационных систем и баз данных
Богданёнка Вячеслава Юрьевича

Руководитель: к.т.н., профессор кафедры ПОиАИС
Макаров Константин Сергеевич

Допустить к защите:

_____ /Макаров К. С./

«25» декабря 2020 г.

Курск, 2020

Содержание

ВВЕДЕНИЕ 3
1 Анализ требований к информационной системе 6
1.1 Описание и анализ предметной области 6
1.2.1 Боковая панель 15
1.2.2 Создание новой учебной комнаты 18
1.2.3 Внутренний интерфейс комнаты в роли преподавателя 19
1.2.4 Интерфейс комнаты пользователя в роли ученика 24
1.3 Анализ функциональных и эксплуатационных требований 25
1.3.1 Стандарты 25
1.3.2 Функциональные требования пользователя 25
1.3.3 Входные данные 26
1.3.4 Выходные данные 26
1.3.5 Требования к интерфейсу 26
1.3.6 Требования к надежности 27
1.3.7 Требования к программной документации 27
1.3.8 Требования к составу и параметрам технических средств 28
1.3.9 Модель вариантов использования 28
1.3.10 Глоссарий проекта 31
1.3.11 Проверка модели на полноту 32
2 Проектирование информационной системы 36
2.1 Разработка архитектуры системы 36
2.2 Разработка модели предметной области 37
2.3 Разработка алгоритма функционирования системы 39
2.4 Проектирование интерфейса пользователя 45
2.5 Реляционная модель данных 47
2.6. Проектирование классов предметной области 50
3 Реализация системы 59
3.1 Реализация программного обеспечения системы 59
3.2 Реализация технического обеспечения 65
ЗАКЛЮЧЕНИЕ 73

ВВЕДЕНИЕ

Разрабатываемый программный продукт является информационной системой обмена файлами и сообщениями для организации образовательного процесса. Целью данного курсового проекта является создание информационной системы, которая сделает возможным организовать и наладить процесс образования в удалённом формате.

Очень важно и необходимо иметь возможность продолжать комфортное обучение в учебном заведении даже при условии, что по каким-то причинам само заведение не может очно принимать учащихся (например при наступлении карантина учебное заведение закрыли на какое-то время). Процесс обучения должен быть непрерывен, а любые ситуации, препятствующие очному посещению учеников или студентов, не должны лишать людей возможности комфортно получать знания и поддерживать контакт с преподавателями.

Программный продукт разрабатывается с целью:

– организации образовательного процесса преподавателем;

– получение учащимися обратной связи от преподавателей по выполненному домашнему заданию;

– налаживание коммуникации между преподавателем и учащимися;

С его помощью упростится процесс обучения студентов в удалённом формате. Студентам не нужно будет связываться с преподавателем в индивидуальном порядке, чтобы получить от него необходимую информацию о процессе обучения или обратную связь по выполненному заданию. Вся информация будет выложена в открытый доступ по ссылке, перейдя по которой учащийся сможет увидеть весь учебный материал и необходимые к выполнению домашние задания. При необходимости учащиеся смогут связаться с преподавателем в личных сообщениях, если у них всё же останутся какие-либо вопросы.

Кроме того, студенту будет доступна вся информация о состоянии выполненных или не выполненных им работ, а также предварительная итоговая

оценка его успеваемости по конкретной дисциплине. Преподавателю такая система тоже поможет, ведь ему не нужно будет в индивидуальном порядке объяснять требования и вести учёт выполненных домашних заданий. Оценка по ним и все файлы выполненных работ будут доступны в любое время с возможностью поиска и сортировки по ФИО учащегося. Также будет иметься возможность автоматического выставления итоговой оценки учащемуся на основании выполненных им работ.

Целью разработки приложения является создание информационной системы обмена файлами и сообщениями для организации образовательного процесса, которая может автоматизировать некоторые рутинные задачи для учеников и преподавателей в процессе обучения или преподавания.

Основными задачами разработки являются:

1. Обеспечение авторизованного входа пользователей в систему для создания или присоединения к учебной комнате.
2. Обеспечение возможности преподавателю создать комнату и наполнить её обучающим материалом.
3. Обеспечению возможности учащимся присоединятся по ссылке к комнате с учебным материалом.
4. Обеспечение возможности преподавателю составлять план сдачи домашних заданий.
5. Обеспечение возможности учащемуся загружать выполненные задания на проверку преподавателю.
6. Возможность преподавателю оценивать выполненные домашние задания.
7. Возможность коммуникации между учащимся и преподавателем в виде отправке друг другу личных сообщений.
8. Возможность автоматического проставления итоговой оценки ученику на основе оценок за выполненные задания.

Для разработки программного продукта применяется среда визуального объектно-ориентированного программирования Microsoft Visual Studio Code. Вместе с СУБД PostgreSQL для создания информационной системы используется CASE-средство Draw.io. Для рассылки по электронной почте используется тестовый SMTP сервер от компании Яндекс.

1 Анализ требований к информационной системе

1.1 Описание и анализ предметной области

На данный момент есть много способов продолжить процесс обучения, если по каким-то причинам само учебное заведение перестало функционировать. Например, существуют сервисы для проведения веб конференций, которые от части смогут заменить учащимся посещение занятий в учебном заведении. Однако вопрос об учебном плане всё равно остаётся открытым для всех.

Предполагается, что учащиеся в случае перевода учебного заведения в удалённый формат могут спросить у преподавателя лично об учебном плане. Однако на практике может случится так, что у учащегося может даже не быть доступа к контактам преподавателя, или же сам преподаватель может находится в отпуске или в больнице, не имея возможность передать группе учащихся или отдельному учащемуся составленный заранее план занятий и необходимые к выполнению задания. Кроме того, развитые страны уже давно применяют на практике дистанционное обучение, при котором все курсовые, лабораторные и прочие работы студент отправляет преподавателю по электронной почте, а приём экзаменов происходит уже в очном формате, чтобы точно убедиться, что всё проходит честно.

Вот несколько примеров по статистике, касающийся дистанционного обучения на 2022 год:

– В среднем во всём мире увеличилось количество платных подписок на сервисы, предоставляющие услуги видео конференций (увеличилось в среднем на 60%, в некоторых странах мира: США, Бельгия, Швейцария, Канада зафиксировано увеличение спроса на 85%)[1]

– В период второй половины 2020го по первую половину 2021го года в развитых странах более 85% учебных заведений перешли на дистанционное обучение. Среди них MIT, Cambridge, МГУ и другие крупнейшие вузы.

[1] https://news.un.org/ru/story/2020/04/1376532

– Как показало исследование НИУ ВШЭ, проведенное в 2019 году, сами преподаватели вузов с ученой степенью невысоко (3,2 балла из 5) оценивают свой уровень владения дистанционными технологиями, а каждый 4-й из них ни разу за последние 3 года не использовал сервисы удаленной видеосвязи для участия в вебинарах и видео-конференциях или проведения аналогичных мероприятий. [2]

– Многочисленные исследования в области образовательных технологий сходятся в том, что в основе онлайн-обучения лежит тщательно спроектированный и спланированный учебный процесс в ЭИОС, поддерживаемый методически обоснованной и целенаправленной последовательностью учебно-методических и контрольно-измерительных материалов, которые обеспечивают достижение результатов обучения в формате исключительно электронного обучения. Ключевым в этом определении является педагогический дизайн, как инструмент проектирования онлайн-курса, что отсутствует в большинстве случаев при резком переходе на "дистант".

Таким образом, учитывая все эти факты возникает довольно странная ситуация. С одной стороны, имеется довольно развитые сервисы, способные заменить в какой-то мере очное обучение студентов (например, Skype, Zoom, или прямой конкурент данной информационной системы – Google Classroom), спрос на которые, как мы узнали только растёт, но с другой стороны многие уже имеющиеся инструменты кажутся преподавателям не интуитивными, непонятными. Людям всё равно не хватает инструмента, помогающего отслеживать сам процесс обучения, будь то количество долгов по домашнему заданию, само домашнее задание, оценки и текущая успеваемость.

Основным вариантом обмена сообщениями и файлами на данный момент является электронная почта. При отправке домашнего задания по электронной почте оно может запросто затеряться среди других писем с абсолютно сторонних сервисов или от посторонних людей. Кроме того, учащийся может иметь адрес электронной почты, по которому преподаватель просто не сможет идентифицировать чью именно работу он получил, если вдруг студент в теме

[2] https://academia.interfax.ru/ru/analytics/research/4491/

письма забудет указать своё ФИО и из какой он группы/класса. Всем участникам диалога становится тяжело отслеживать, когда и в какое время было отправлено домашнее задание, от кого оно было отправлено и было ли оно в итоге проверено и получена за него оценка.

Все эти недостатки можно устранить, создав ИС, которая не только исправит недостатки предшественников, но и привнесёт понятные и удобные нововведения.

Данная система позволит:

– Организовать образовательный процесс

– Отслеживать статистику о процессе обучения

– Собрать все необходимые материалы для получения знаний по определённой дисциплине в одном месте для удобного пользования

– Идентифицировать каждого участника образовательного процесса, помимо ФИО выводить для преподавателя основную информацию по его успеваемости

– Сделать удобным и комфортным процесс обучения в дистанционном формате путём возможности постоянного поддерживания связи преподавателя с учениками

Основное взаимодействие между системой и пользователем будет происходить в рамках комнат (classroom) связанных с какой-либо конкретной учебной дисциплиной. Возможности пользователя будут зависеть от того, в какой роли в комнате он находится. Если он создал комнату, ему по умолчанию присваивается звание создателя (host) в этой комнате, если же он вступил в уже созданную, то будет иметь права участника (participant). В зависимости от этого он будет иметь разный набор возможностей в рамках одной комнаты. Однако каждый пользователь системы сможет:

– Регистрироваться в сервисе, указав своё ФИО и основные данные для связи в случае необходимости

– Авторизоваться в сервисе и изменять данные, заполненные при регистрации

– Просматривать профили участников какой-либо комнаты и видеть основную информацию, которые они указали

– Создавать отдельную комнату (classroom) с учебной дисциплиной

– Вступать в уже созданные комнаты переходя по ссылке

Возможности пользователя с ролью преподаватель:

– Выкладывать новые учебные материалы (файлы документов и пояснения к ним)

– Добавлять в созданную комнату новых участников для участия в процессе обучения

– Назначать участникам роли (например, если преподавателей может быть несколько)

– Выкладывать домашнее задание со сроком сдачи

– Оценивать выполненные домашние задания

– Отвечать на личные сообщения учеников

Возможности пользователя с ролью участника:

– Просматривать и комментировать учебный материал

– Вовремя выполнять и отправлять преподавателю домашние задания

– Начать диалог с преподавателем в личных сообщениях

При работе с данной системой преподавателю сперва будет необходимо пройти регистрации и авторизацию в сервисе. Далее ему нужно будет создать новую комнату с названием его предмета. После чего загрузить туда учебный материал с названием и кратким описанием его дисциплины, прикрепить файлы с документами по необходимости. Также преподаватель может создать домашние задания, которые ученикам необходимо будет выполнить к какому-то конкретному сроку. После чего ему надо будет раздать ссылку на вступление в

комнату всем участникам. При необходимости преподаватель может в любое время дополнить, или изменить учебный материал или домашние работы. Как только ученики пришлют работы на проверку он сможет оценить их или прокомментировать, если у него останутся какие-либо вопросы к выполненным работам.

Сами учащиеся же в данной системе будут иметь возможность не только задавать вопросы преподавателю по конкретному материалу, но и иметь возможность связаться с ним через личные сообщения.

Каждый пользователь после регистрации сможет поменять свои учётные данные, например, сменить email, пароль, ФИО или номер телефона.

Преподавателю будет доступна помимо основной информации ещё и статистика по выполненным работам всех студентов и предварительная итоговая оценка обучающегося. Обычным участникам комнаты будет доступна только статистика по своим работам в конкретной комнате.

Для реализации проекта помимо подбора окружения для разработки необходимо будет также настроить SMTP сервер для рассылки оповещений пользователям по почте.

Simple Mail Transfer Protocol (SMTP) — простой протокол связи, применяемый с целью пересылки электронных писем с сервера отправителя на сервер получателя. Этот протокол не рассчитан на обработку входящих сообщений, его используют для отправки и последующей доставки писем адресату. Преимущественно с помощью SMTP отправляют массовые и транзакционные рассылки.[3]

Также SMTP сервер понадобиться при подтверждении электронной почты, чтобы удостовериться, что почта, указанная при регистрации, точно принадлежит пользователю ему надо будет перейти в указанный почтовый ящик и перейти по ссылке в письме. После чего, его учётная запись активируется и, он сможет пользоваться системой и получать уведомления о событиях, происходящих в комнатах.

[3] https://www.unisender.com/ru/support/about/glossary/chto-takoe-smtp/

Само собой, систему нужно будет связать с базой данных, в которой будет хранится информация о данных пользователя, комнатах, и участниках комнат (их роли, дата присоединения и прочее), а также выполненные домашние задания и учебные материалы, которые будут выводится в конечный интерфейс.

На данный момент в основном приложения представлены в двух видах: десктопный и веб приложения.

Десктопное приложение — программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы. Приложение такого вида требует установки на каждое устройство, с которого его планируется запускать. К десктопным приложениям также относятся и мобильные приложения, хотя некоторые эксперты выделяют их как отдельную категорию, всё же между дескотпным и мобильным приложением разница стоит лишь в том, на каком устройстве их запускают. По своей сути и принципам работы они совершенно одинаковы. Как правило это высоконагруженные приложения для работы с устройством напрямую (например, работа с принтерами, драйверами устройств и прочее).[4]

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.[5]

Для разработки конкретно данной системы был выбран веб интерфейс. Так как он позволит взаимодействовать с ним конечному пользователю с любого устройства, поддерживающего подключение к интернету в любое время. Это может быть особенно полезно, если, например, нужно быстро ответить преподавателю или загрузить выполненное задание с любого устройства на котором есть интернет, войдя в систему под своими данными.

---

[4] https://freematiq.com/uslugi/desktop-prilozheniya/
[5] https://www.azoft.ru/blog/web-apps/

На рисунке 1 представлена контекстная DFD-диаграмма системы обмена файлами и сообщениями для организации образовательного процесса.

Рисунок 1 - контекстная DFD-диаграмма системы обмена файлами и сообщениями для организации образовательного процесса

На рисунке 2 приведено описание бизнес-процесса в BPMN.

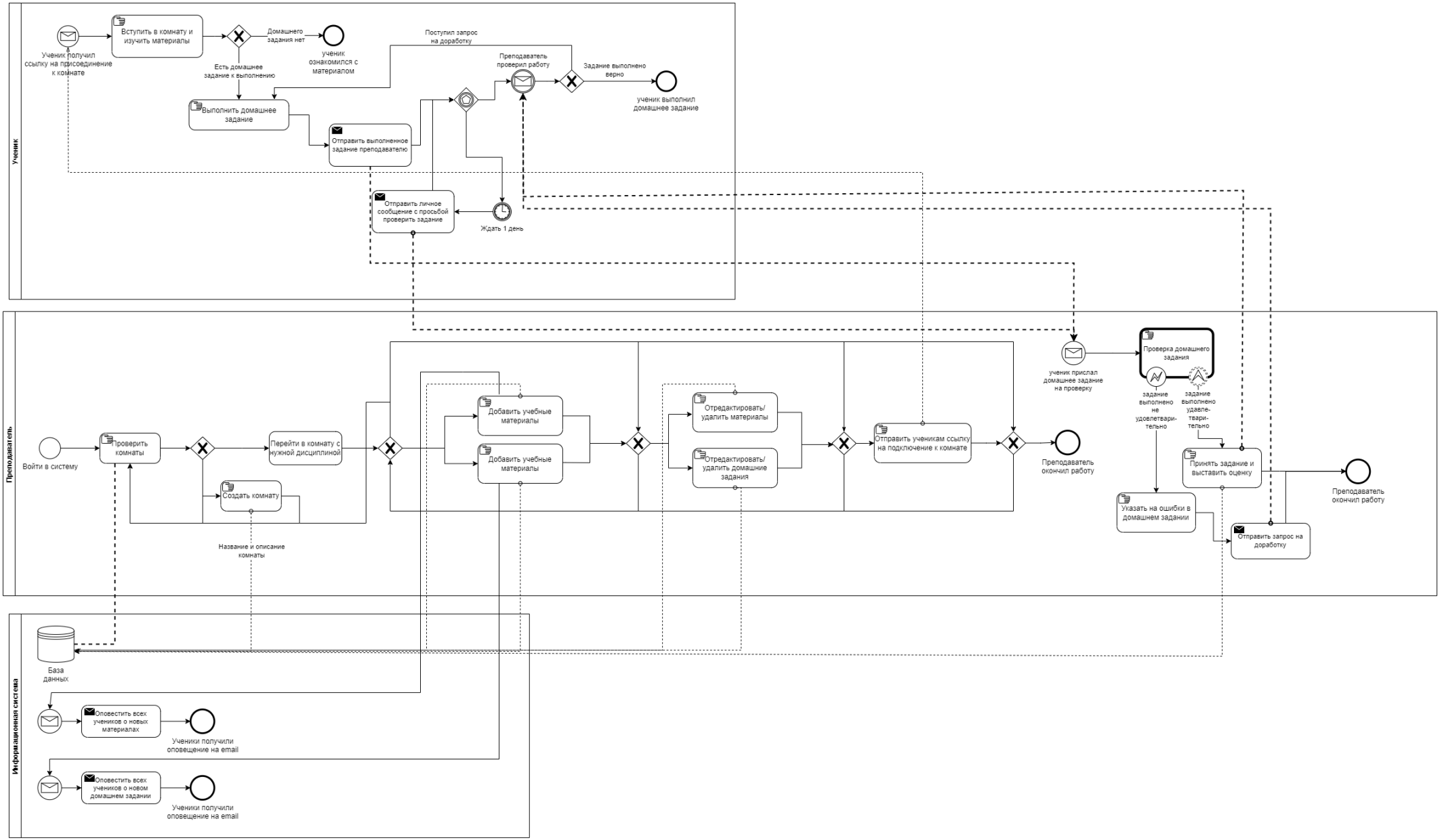


Рисунок 2 - описание бизнес-процесса в BPMN

1.2 Обзор и анализ возможных альтернатив

При поиске альтернативных вариантов разрабатываемой информационной системы была найдена следующая информационная система кинообслуживания [1].

Google Класс — бесплатный веб-сервис, разработанный Google для школ, который призван упростить создание, распространение и оценку заданий безбумажным способом. Основная цель Google Класс — упростить процесс обмена файлами между учителями и учениками[1].[6]

Рабочее место преподавателя обеспечивает создание, редактирование и контроль по таким объектам как:

— комнаты;

— материалы;

— домашние задания;

— комментарии;

— участники комнаты;

После перехода на сайт https://classroom.google.com/u/0/h если пользователь вошёл в свой гугл аккаунт будет выведена следующая страница, представленная на рисунке 3.

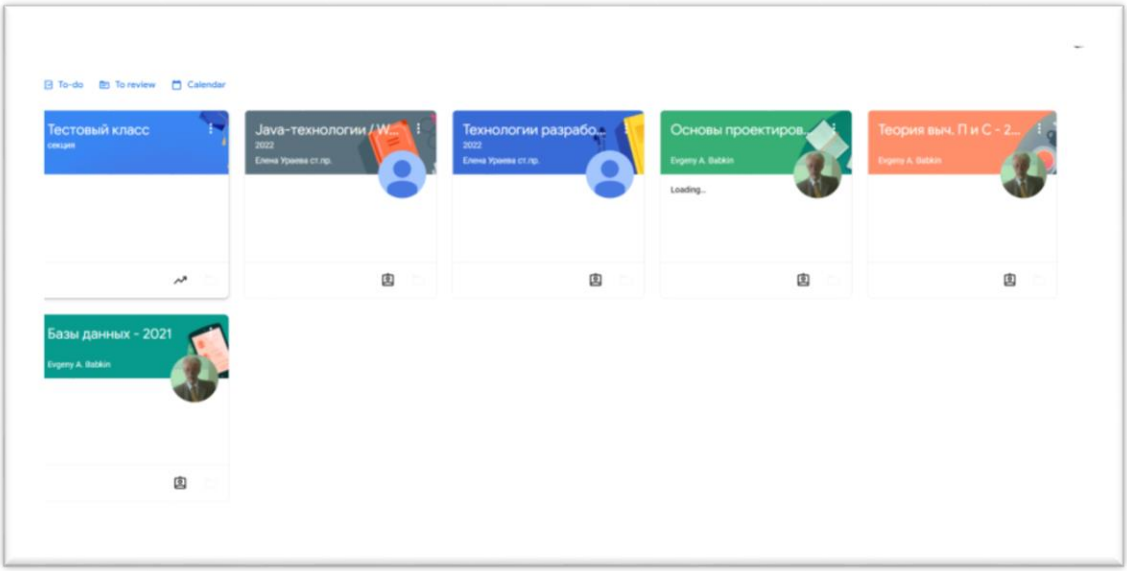


Рисунок 3 – Главная страница Google Classroom

[6] https://classroom.google.com/

1.2.1 Боковая панель

В верхней части экрана находится шапка с названием сервиса. Если нажать на три полоски в верхнем левом углу, откроется боковая панель сайта с доступными действиями, представленное на рисунке 4.

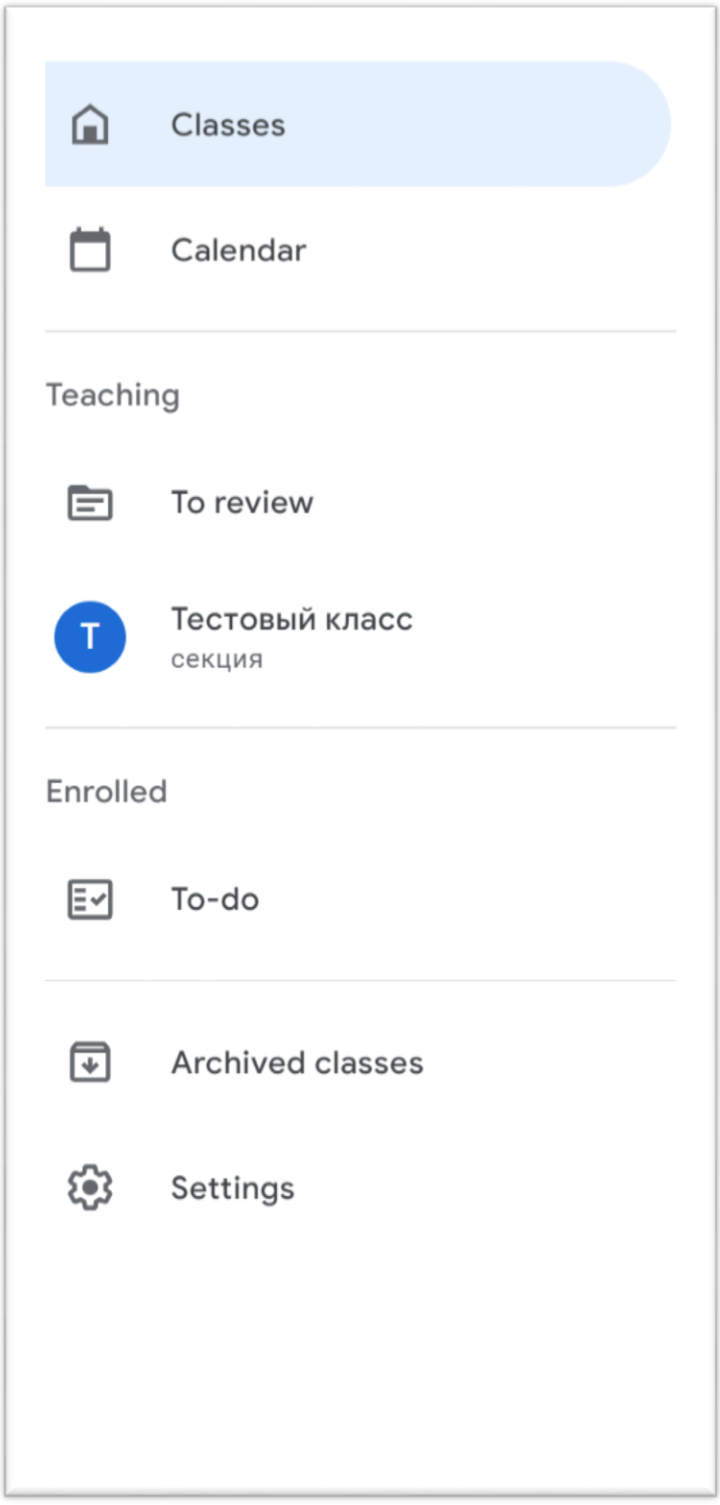


Рисунок 4 – открывающееся меню на главной странице

Во вкладке Teaching будут выводиться комнаты, в которых пользователь участвует в роли преподавателя. Если нажать кнопку «To review» откроется список выполненных учениками домашних работ, которые ждут проверки. Экран представлен на рисунке 5.

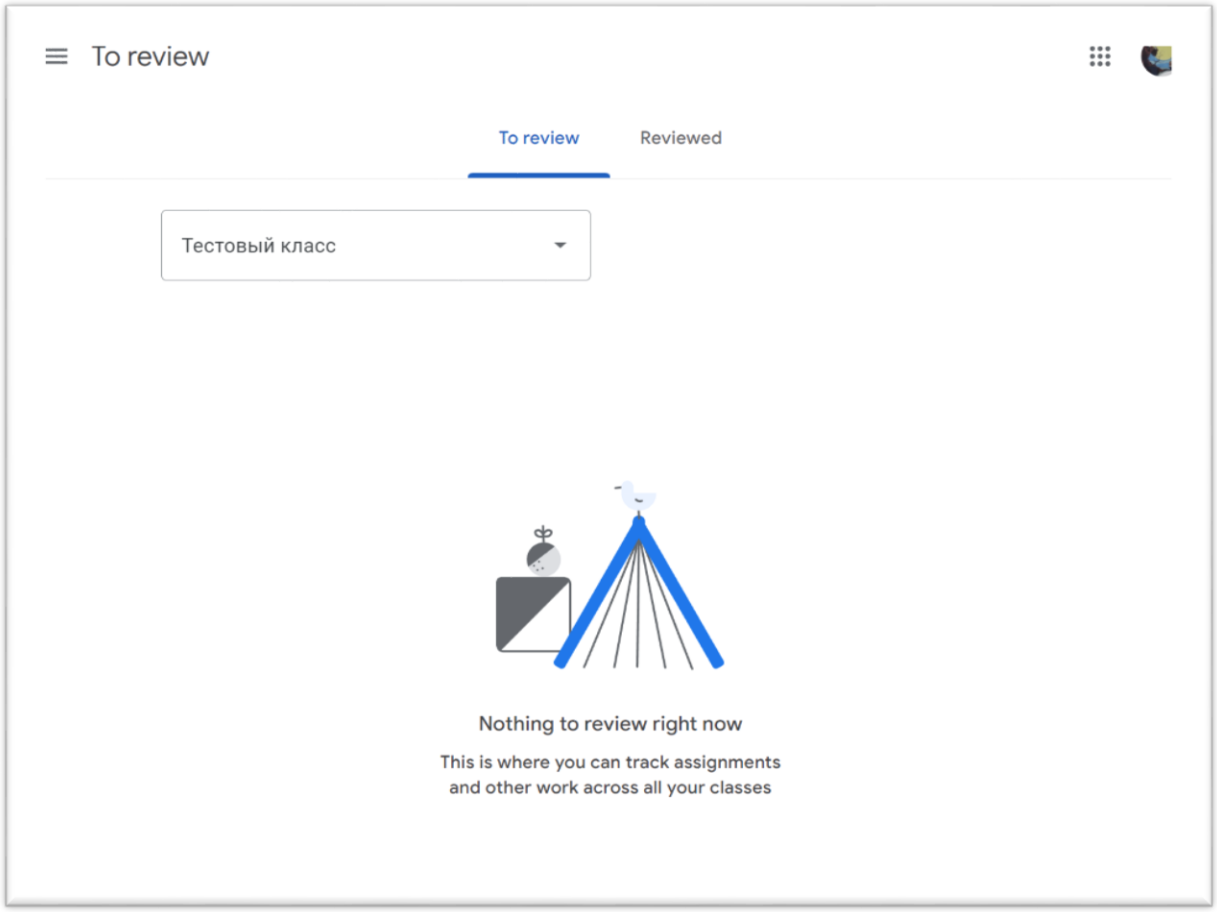


Рисунок 5 – переход по ссылке «To review» с главной страницы из открывающегося меню

Во вкладке enrolled будут выведены комнаты, в которых вы участвуете в роли ученика. Если нажать на пункт «To do» откроется список домашних заданий, которые пользователю необходимо выполнить. Переход на страницу представлен на рисунке 6.

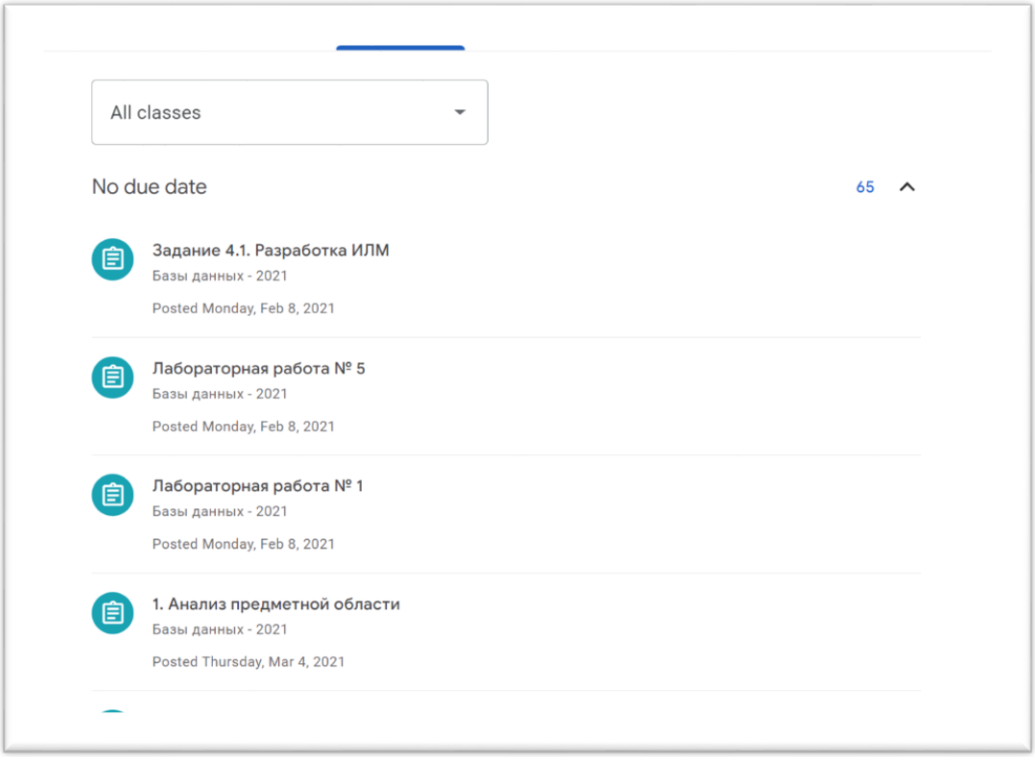


Рисунок 6 – переход по ссылке «To do» с главной страницы из открывающегося меню

Представлена также кнопка «Archived classes» перейдя по которой можно перейти в архив комнат. Комната считается заархивированной, если преподаватель решит, что обучение по данной дисциплине завершено и удалит комнату. Внешний вид страницы с архивными комнатами приведён на рисунке 7.

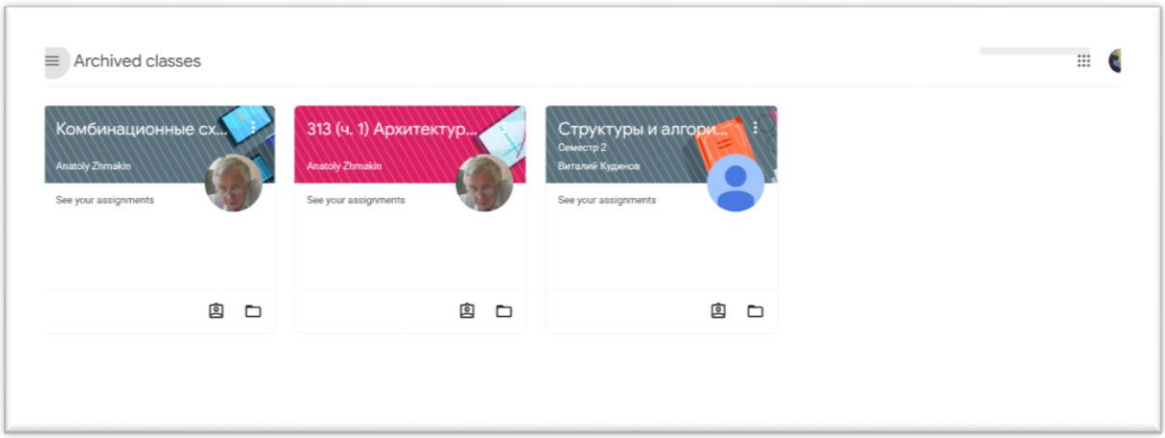


Рисунок 7 – Внешний вид страницы с архивными комнатами

При нажатии на кнопку «Calendar» пользователь попадает на календарь с записью его событий. Например, проверка или выполнение домашних работ. Внешний вид страницы при переходе в календарь представлены на рисунке 8.

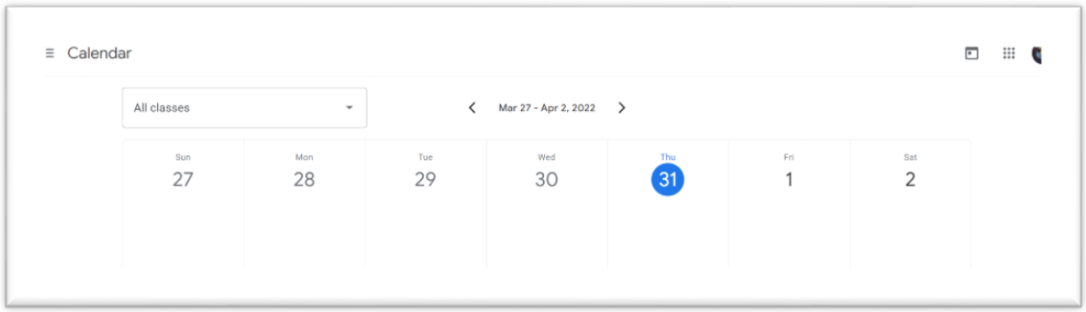


Рисунок 8 - Внешний вид страницы при переходе в календарь

1.2.2 Создание новой учебной комнаты

Помимо выдвигающегося меню в шапке также присутствует кнопка просмотра перехода на другие сервисы от компании Google и кнопка «+», нажав на которую и выбрав пункт Create можно создать новую комнату. Перед созданием комнаты вас попросят принять пользовательское соглашение. Если принять его откроется всплывающее окно с созданием комнаты. Основная информация о нём. Форма создания класса представлена на рисунке 9.

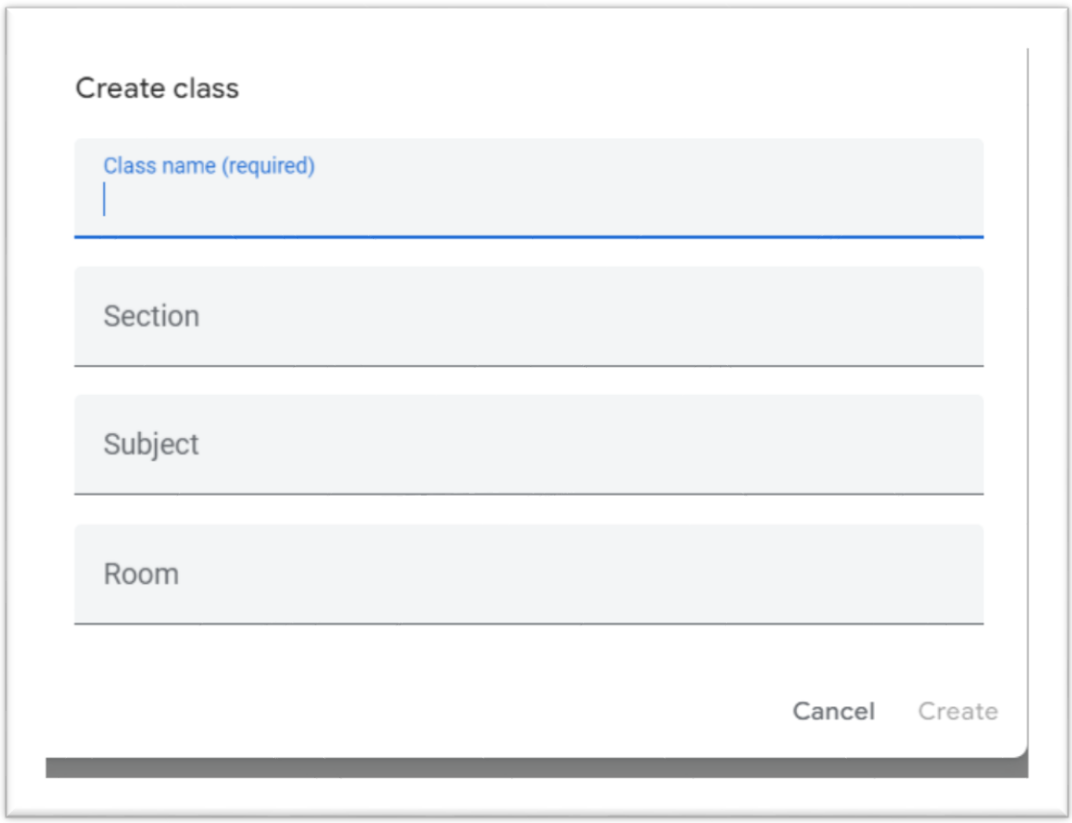


Рисунок 9 - Форма создания класса

Заполнив поля, пользователя перенаправит в только что созданную комнату.

1.2.3 Внутренний интерфейс комнаты в роли преподавателя

В созданной комнате или же классе, пользователь будет находится в роли преподавателя. Он сможет добавлять, обновлять и редактировать материалы, домашние работы, добавлять новых участников вручную и т.д. На рисунке 10 представлена лента материалов комнаты, которые загрузил преподаватель, а также комментарии к ней.

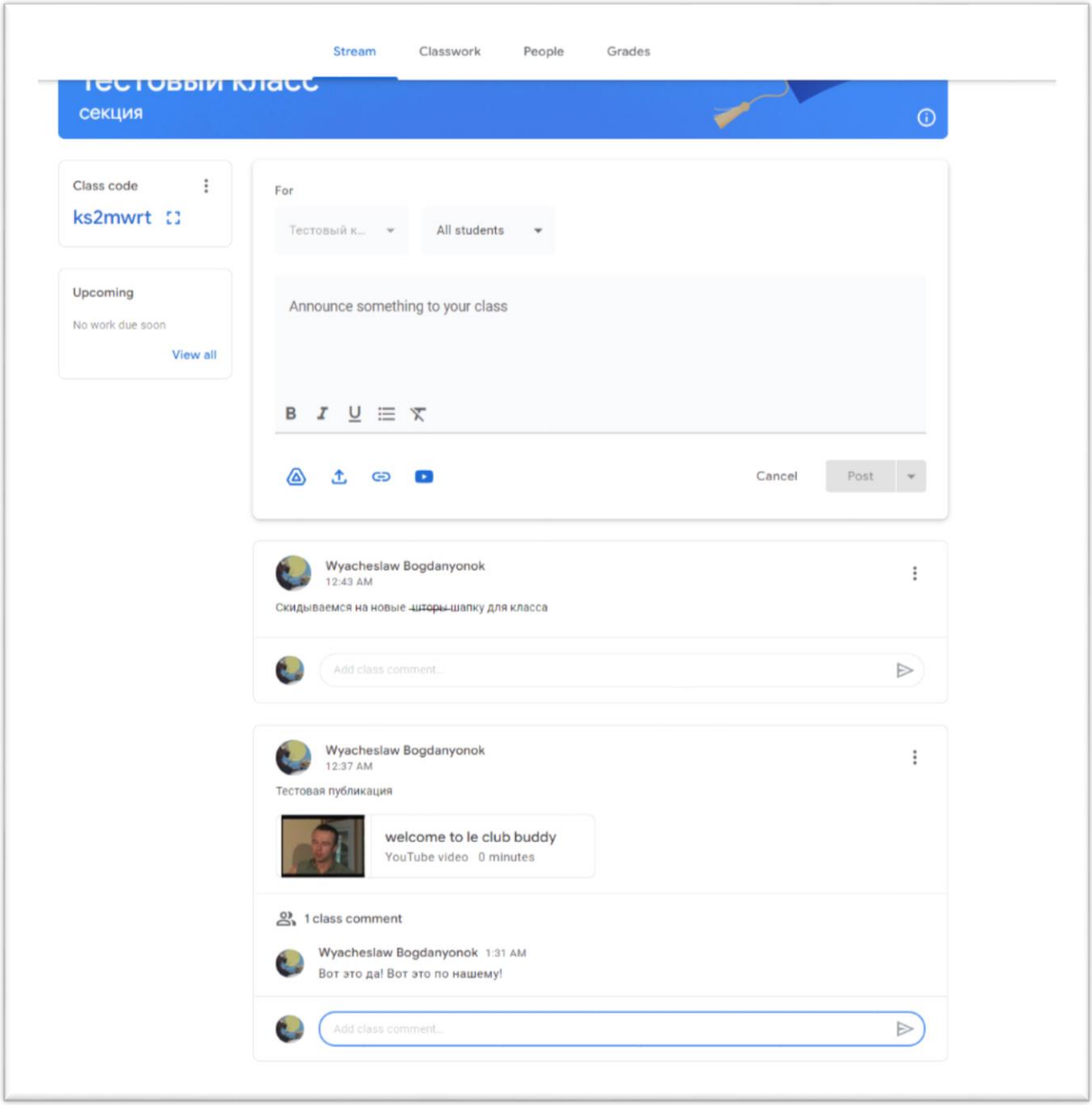


Рисунок 10 – Лента материалов комнаты, которые загрузил преподаватель, а также комментарии к ней

При создании новых материалов будет показана форма. В которой можно написать название файла, а также прикрепить файлы с компьютера или из файлового хранилища Google Drive, а также прикрепить ссылку или видео с платформы YouTube. Форма добавления новых материалов в ленту класса представлена на рисунке 11.

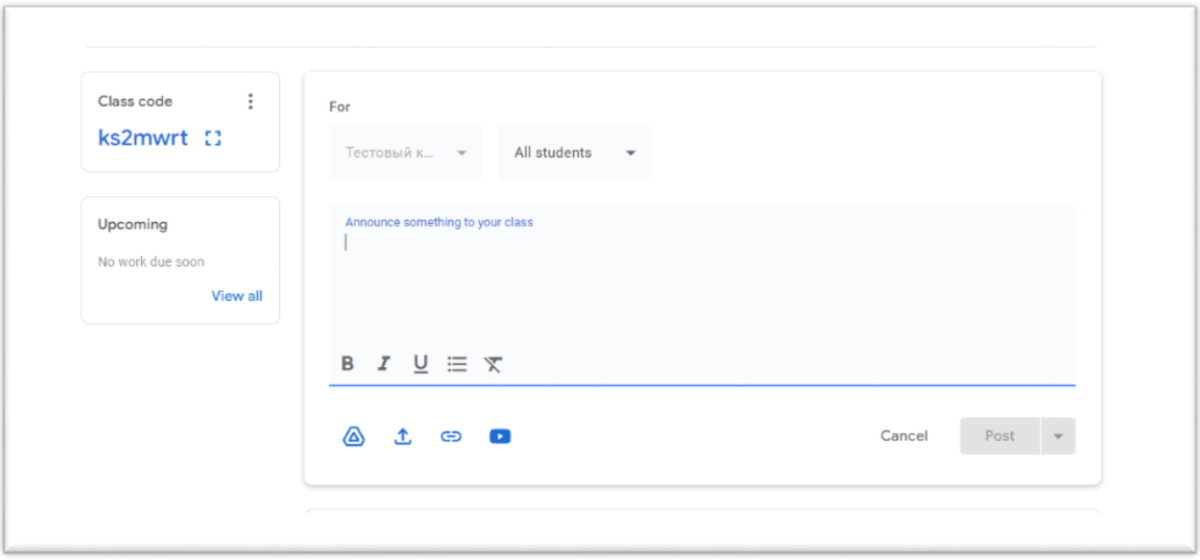


Рисунок 11 - Форма добавления новых материалов в ленту класса

Помимо основной ленты в комнате также присутствуют и другие вкладки. «Classwork» - вкладка в которой преподаватель может проверить или выложить для выполнения домашние работы, которые он может поставить как со сроком, так и без срока сдачи. Вкладка «People» в которой преподаватель может добавить студентов или других преподавателей вручную и «Grades» - вкладка с оценками, в которой можно настроить систему оценивания и сами оценки учеников. Внешние виды страниц с данными вкладками представлены на рисунках 11-15.

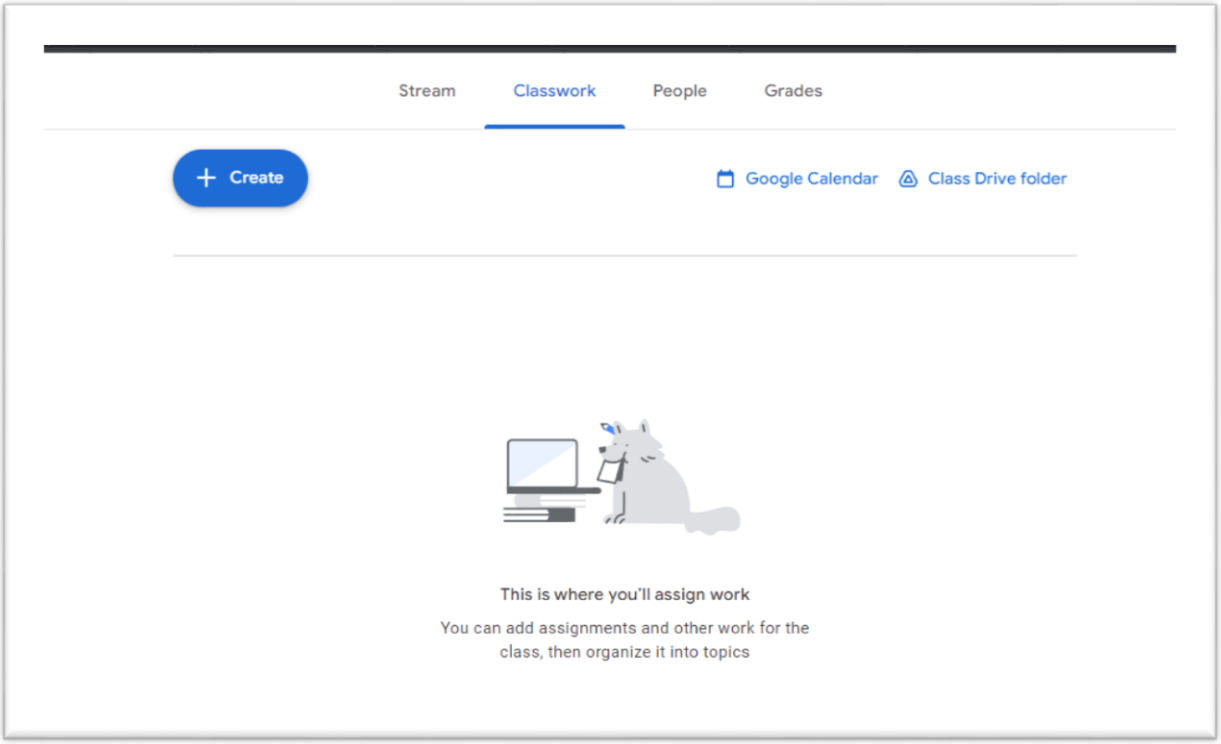


Рисунок 11 – Внешний вид вкладки «Classwork»

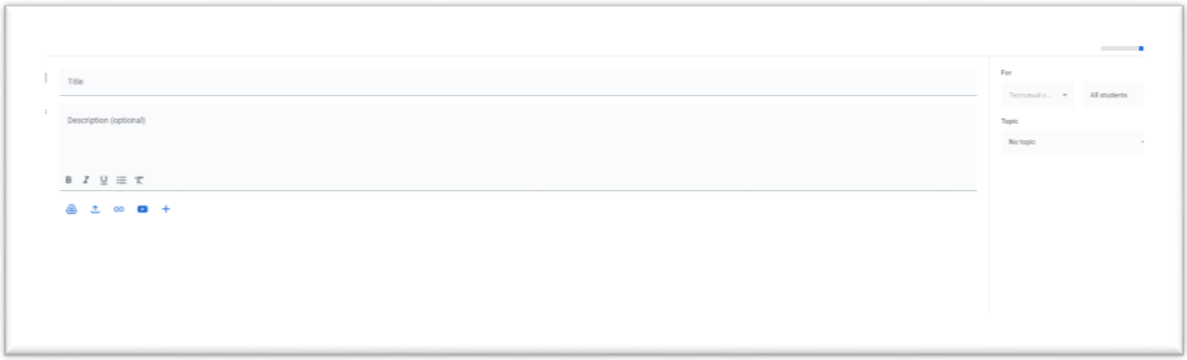


Рисунок 12 – Внешний вид формы создания новой домашней работы

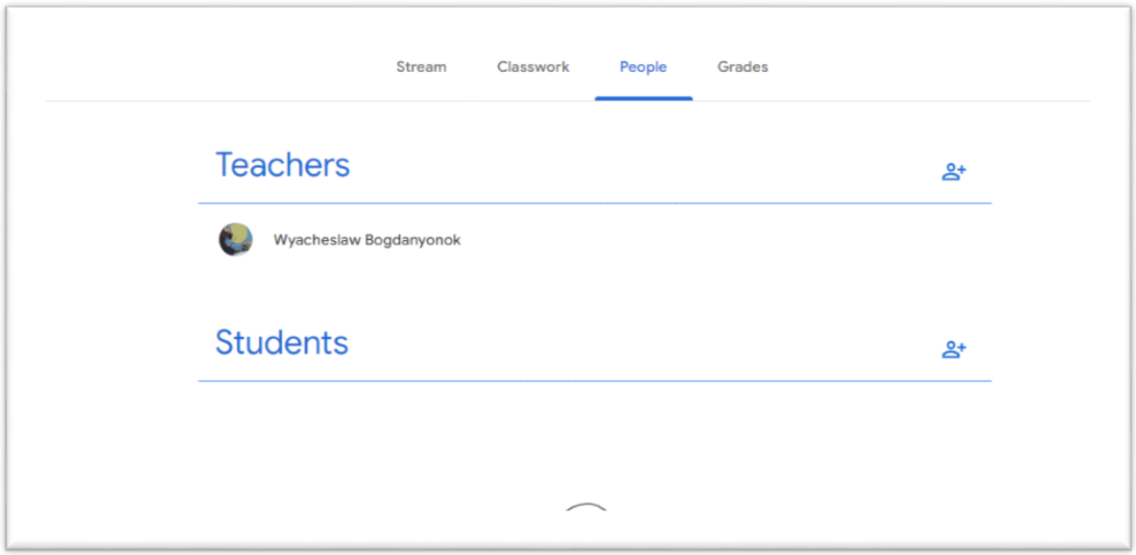


Рисунок 13 – Внешний вид вкладки «People»

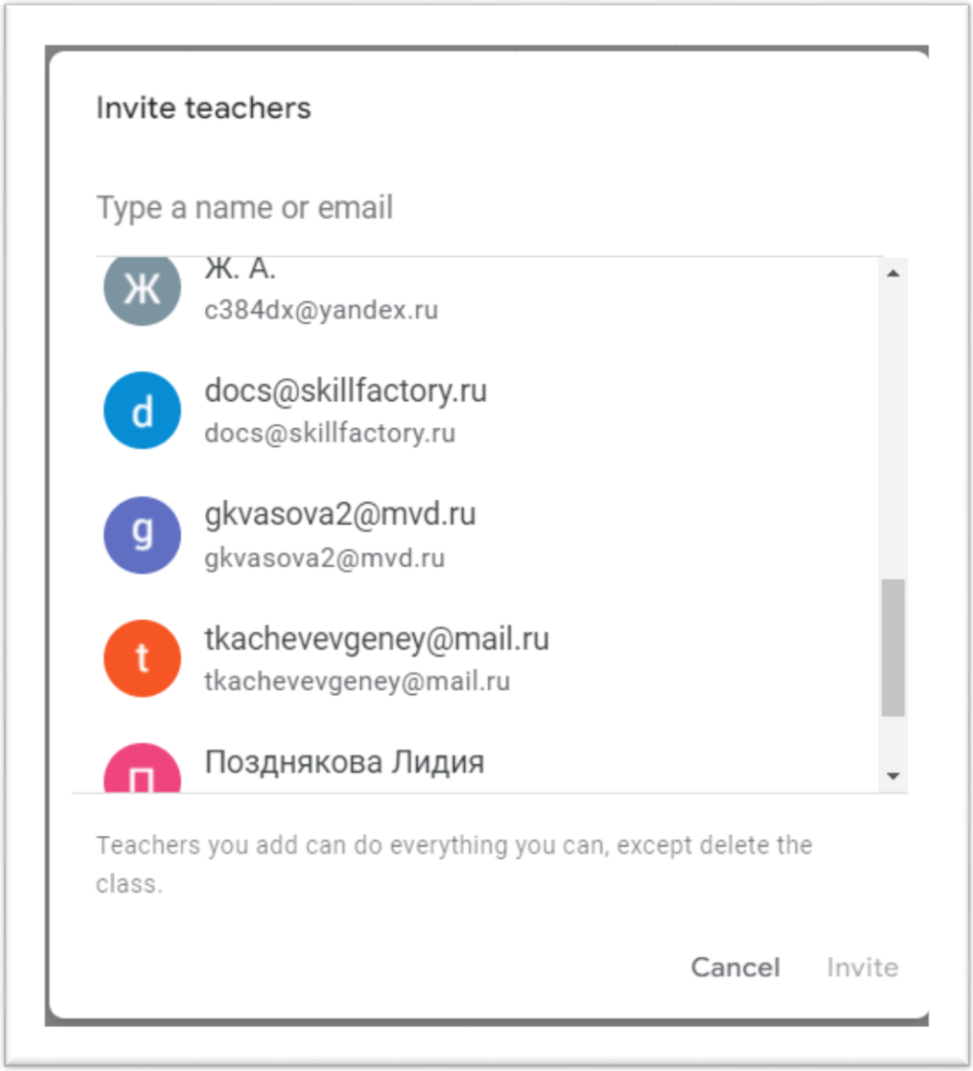


Рисунок 14 – Внешний вид формы добавления новых участников в комнату

1.2.4 Интерфейс комнаты пользователя в роли ученика

Интерфейс пользователя в комнате с ролью ученика не имеет между собой существенных отличий от комнаты, в которой он находится в роли преподавателя. Интерфейс сохраняется тот же, но нет доступа к вкладке «Grades» и в зависимости от настроек комнаты пользователь может не иметь возможности приглашать других пользователей в комнату и оставлять комментарии или добавлять новые записи. Внешний вид интерфейса комнаты с ролью пользователя представлен на рисунке 17.

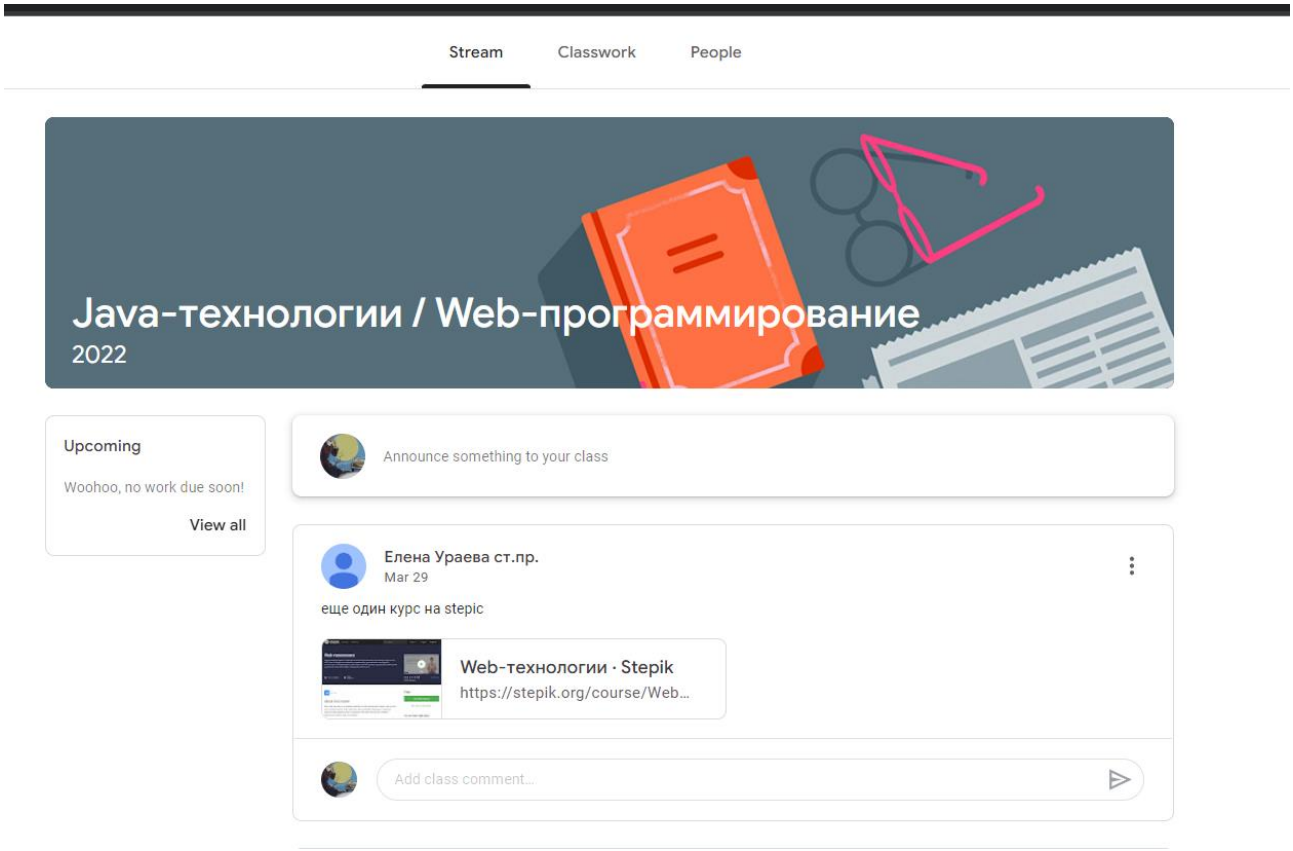


Рисунок 17 - Внешний вид интерфейса комнаты в роли ученика

В роли ученика у пользователя также появляется возможность отправлять преподавателю домашние работы на проверку. Для этого необходимо перейти во вкладку «Classwork» и нажать на материал с домашним заданием, которое ученик выполнил и готов загрузить преподавателю на проверку. Внешний вид формы загрузки домашнего задания на проверку преподавателю представлен на рисунке 18.

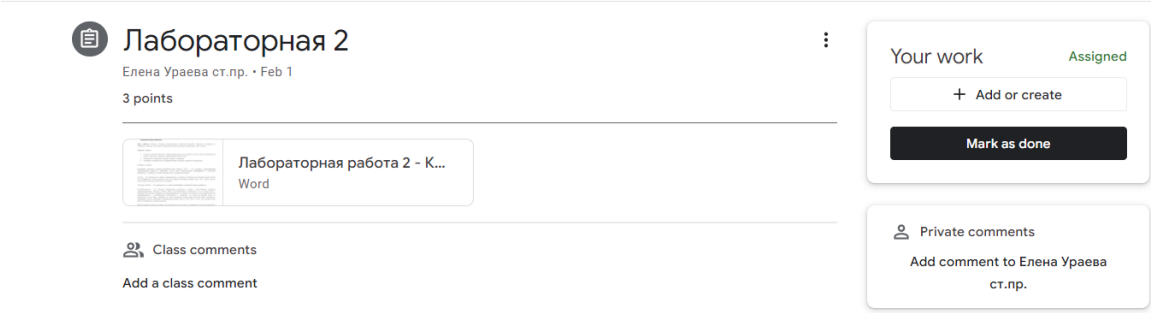


Рисунок 18 - Внешний вид формы загрузки домашнего задания на проверку преподавателю

1.3 Анализ функциональных и эксплуатационных требований

1.3.1 Стандарты

Программный продукт разрабатывается на основании следующих государственных стандартов:

1. Межгосударственный стандарт ГОСТ 7.32-2017 «Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно – исследовательской работе. Структура и правила оформления».

2. Международный стандарт ISO/IEC 12207. Информационные технологии. Процессы жизненного цикла программного обеспечения.

3. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания.

4. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

5. ГОСТ 34.603-92. Информационная технология. Виды испытаний автоматизированных систем.

1.3.2 Функциональные требования пользователя

Программный продукт, разрабатываемый в рамках курсового проекта должен удовлетворять следующему перечню функциональных требований:

— разграничение ролей в системе на пользователя и преподавателя;

— обеспечивать удобство образовательного процесса для студентов и преподавателей;

— обмен файлами между преподавателями и учениками;

— обмен сообщениями между преподавателями и учениками;

— оценивание успеваемости студентов;

1.3.3 Входные данные

Входными данными при работе программы будут являться учётные данные пользователя, комментарии к записям в комнате, личные сообщения, а также документы в .docx и .pdf формате, таблицы и презентации.

1.3.4 Выходные данные

Выходными данными при работе программы являются данные, выводимые на экран пользователя. Статистика по успеваемости, личные сообщения и файлы, отправленные другими пользователями. Язык выходных данных в приложении – русский.

1.3.5 Требования к интерфейсу

Программный продукт должен содержать форму авторизации и регистрации пользователей, переход на которую будет осуществлять с главной страницы приложения.

Авторизованному пользователю должны быть доступны комнаты, в которых он является участником и имеет роль ученика или преподавателя. В этих комнатах пользователь может переходить по вкладкам «Учебные материалы», «Домашнее задание», «Участники» и «Статистика успеваемости». В каждой вкладке отображается соответствующий материал. Необходимо реализовать эргономичный интерфейс со списком карточек с краткой информацией о каждом элементе и возможностью преподавателя добавлять, изменять и редактировать выложенные учебные материалы и домашние задания, а также при необходимости удалять участников из комнаты.

Для любого из элементов домашнего задания или учебного материала преподаватель может прикреплять файлы при необходимости. При отображении элементов, если к нему прикреплены файлы пользователь при нажатии на них должен иметь возможность загрузить их к себе на компьютер через браузер.

На каждой странице со списками необходимо будет реализовать фильтры и сортировки по основным параметрам, чтобы облегчить на них поиск информации.

Каждый пользователь может просматривать полную информацию об учебных материалах, домашних заданиях (если он является участником комнаты, за которой они закреплены), а также основную информацию об участниках группы в которой он находится.

Интерфейс веб сервиса должен быть практичным и интуитивно понятным. Иметь не навязчивую, сдержанную цветовую гамму, чтобы не привлекать внимание и отвлекать от процесса обучения. Так же сервис должен иметь так называемую адаптивность – свойство, при котором сайт одинаково хорошо смотрится на устройстве с любым размером экрана и разрешением.

1.3.6 Требования к надежности

При работе с программным продуктом необходимо предусмотреть:

— контроль вводимой пользователем информации;

— возможность отслеживания ошибок, допускаемых пользователем, и последующей реакции программы на них;

— предотвратить возможность злоумышленников получать учётные данные пользователей;

— обеспечивать по возможности отказоустойчивость при избыточной нагрузке на сервер с приложением;

— предусмотреть блокировку некорректных действий пользователя при работе с системой;

1.3.7 Требования к программной документации

В состав сопровождающей документации программного продукта должны входить следующие компоненты:

1. Пояснительная записка на 30 – 50 листах, содержащая описание разработки.

2. В приложение к пояснительной записке предоставить исходные файлы проектов, использованные для построения всех диаграмм.

3. К пояснительной записке также приложить исходный код программы на CD диске с инструкцией по запуску программы и развёртыванию её на любом сервере.

1.3.8 Требования к составу и параметрам технических средств

Поскольку данная ИС представляет из себя веб приложения, то можно разделить требования к техническим средствам на два пункта: технические требования для клиента (конечный пользователь, зашедший на страницу сервиса в браузере) и технические требования к серверу (устройству, на котором запущен обработчик запросов от пользователя и выполняется основной исходный код программы, перед отправкой ответов пользователю).

Технические требования к серверу:

— Тип процессора Pentium; минимум 8 ядер;

— Объём оперативно запоминающего устройства – не менее 16 гб;

— Стабильное и непрерывное подключение к интернету с пропускной способностью не менее 100мб/сек;

Технические требования к клиенту:

— Процессор с количеством ядер более 2х.

— Объём оперативной памяти – 512 мб.

— Любое устройство вывода графики (монитор, экран планшета или смартфона)

1.3.9 Модель вариантов использования

1.3.9.1 Диаграмма вариантов использования

Действующие лица для диаграммы вариантов использования приведены в таблице 1.

Таблица 1 – Действующие лица

Термин	Значение
Преподаватель	Лицо, организующее образовательный процесс и обеспечивающее проверку работ учеников.
Ученик	Лицо, проходящее обучение по какой-либо дисциплине, организованной преподавателем.

На основании анализа требований пользователя были выделены следующие варианты использования, представленные в таблице 2.

Таблица 2 – Описание вариантов использования

Термин	Значение
Выложить учебные материалы	Загрузить в систему информацию, на основе которой будет происходить проверка знаний учеников
Выложить домашнее задание	Загрузить в систему файлы и текст пояснения с заданием, которое предстоит выполнить ученикам
Изучить материалы	Ознакомиться с учебными материалами и усвоить информацию в них
Выполнить домашнее задание	Выполнить домашнее задание приложенное преподавателем в соответствии с требованиями
Проверить домашнее задание	Выполнить проверку домашнего задания, загруженного учеником
Выставить оценки	Выставить оценку за выполненное домашнее задание

На основании всех выше рассмотренных вариантов использования была составлена диаграмма вариантов использования, представленная на рисунке 16.

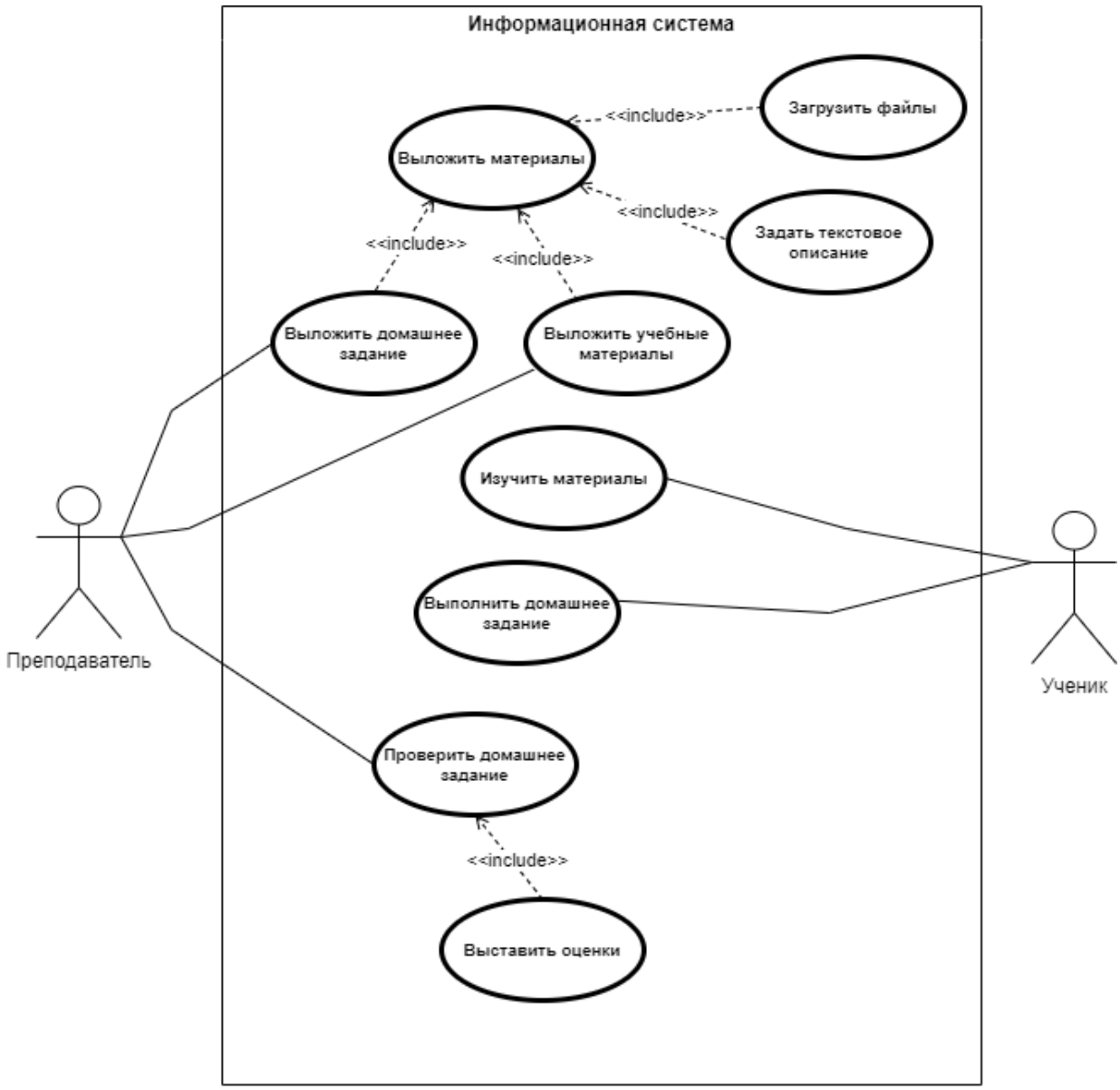


Рисунок 16 - Диаграмма вариантов использования

1.3.9.2 Описание варианта использования «Проверить домашнее задание»

Действующие лица: преподаватель.

Заинтересованные лица и их требования:

— Преподаватель выставил домашнее задание к выполнению;

— Преподаватель хочет оценить правильность выполнения домашнего задания;

— Преподаватель хочет выставить оценку за домашнее задание;

Предусловия: Пользователь должен быть авторизован в системе; Пользователь должен войти в созданную им комнату; Пользователь должен был

загрузить домашнее задание к выполнению; Ученик, находящийся в этой комнате, должен прислать задание на проверку преподавателю;

Постусловие: Если работа выполнена верно, то преподаватель выставляет оценку и закрывает выполнение этой домашнее работы для ученика, иначе – отправляет домашнее задание на доработку.

Основной сценарий:

1. Система отображает выполненные домашние задания.
2. Пользователь выбирает нужное домашнее задание для проверки.
3. Система отображает файлы и описание приложенной выполненной работы.
4. Система предлагает оценить правильность выполнения работы или же отклонить её и отправить на доработку.
5. Преподаватель вводит оценку и комментарий (по желанию) и нажимает принять.
6. Система помечает данную работку как выполненную.

Альтернативные потоки:

1а. Если нет выполненных домашних заданий, система отображает надпись: «Нет выполненных домашних заданий».

5а. Если преподаватель отправляет домашнее задание на доработку, то система помечает домашнюю работу не выполненной и отправляет обратно ученику.

1.3.10 Глоссарий проекта

В таблице 3 приведены термины предметной области и их значения.

Таблица 3 – термины и их значения

Термин	Значение
Комната	Некоторое лобби, в рамках которого проходит весь образовательный процесс. А конкретно: выкладываются домашние задания, учебный материал, проверяются выполненные работы и

	происходит процесс общения преподавателя с учениками.
Преподаватель	Пользователь создавший комнату, имеющий права на любые действия с материалом внутри комнаты и с самой комнатой, вплоть до переименования и удаления её.
Ученик	Пользователь, вступивший в комнату. Имеет права выкладывать домашнее задание на проверку преподавателю, оставлять сообщения в комнате, редактировать и удалять их.
Учебный материал	Определённая текстовая и файловая информация. Не требует никаких действий с ней, кроме изучения.
Домашняя работа	Тоже самое, что и учебный материал, но требует выполнения в назначенный срок.
Выполненная домашняя работа	Определённая файловая и текстовая информация, прикреплённая к домашней работе, требующая проверки преподавателя.

1.3.11 Проверка модели на полноту

Проверка на полноту диаграммы вариантов использования производится по операциям, выполняемым над основными объектами (табл. 4).

Основными объектами, упоминаемыми в модели вариантов использования являются: учебный материал, домашнее задание, выполненное домашнее задание, комната.

Таблица 4 – Проверка на полноту

Варианты использования	Объекты			
	Учебный материал	Домашнее задание	Выполненное домашнее задание	Комната
Добавление материалов	1, 2, 3, 4	1, 2, 3, 4		
Проверка домашнего задания			1, 2, 3	
Управление комнатой				1, 2, 3, 4
Выставление оценок			1	
Составление статистики об оценках учеников			2	

В таблице 4 обозначены виды операций:

1 – создание;

2 – просмотр;

3 – изменение;

4 – удаление.

Над объектом «Проверенное домашнее задание» нет операции удаления (3) ни в одном варианте использования, над объектом «Выставление оценок» можно совершать только создание (1), над объектом «Составление статистики» об оценках учеников можно осуществлять только просмотр.

Операция создания и редактирования выполненного домашнего задания осуществляются только действующим лицом «ученик». Операции создания (1), удаления (4) и изменения (3) над учебными материалами и домашним заданием производит лишь действующее лицо «преподаватель. Создание выставления оценок производит только действующее лицо «преподаватель».

Таблица 5 – Анализ полноты выполнения требования пользователя

Требования пользователя	**Варианты использования**				
	Управление комнатой	Добавление материалов	Проверка домашнего задания	Выставление оценок	Составление статистики об оценках учеников
Обмен файлами между преподавателями и учениками		+	+		
Обеспечивать удобство образовательного процесса для студентов и преподавателей			+	+	+
Обмен сообщениями между преподавателями и учениками			+		
Оценивание успеваемости студентов				+	+
Различение ролей в системе	+	+			

2 Проектирование информационной системы

2.1 Разработка архитектуры системы

Разрабатываемое приложение является клиент-серверным приложением.

Основным требованием к использованию данного приложения является наличие персонального устройства, поддерживающего работу с современными браузерами и имеющими стабильный доступ к интернету. Кроме того, для регистрации пользователю понадобится иметь персональный электронный почтовый ящик и иметь к нему доступ, чтобы подтвердить свою регистрацию и получать оповещения при работе с системой.

На рисунке 17 приведена предварительная диаграмма развертывания разрабатываемого приложения – архитектура технических средств системы.

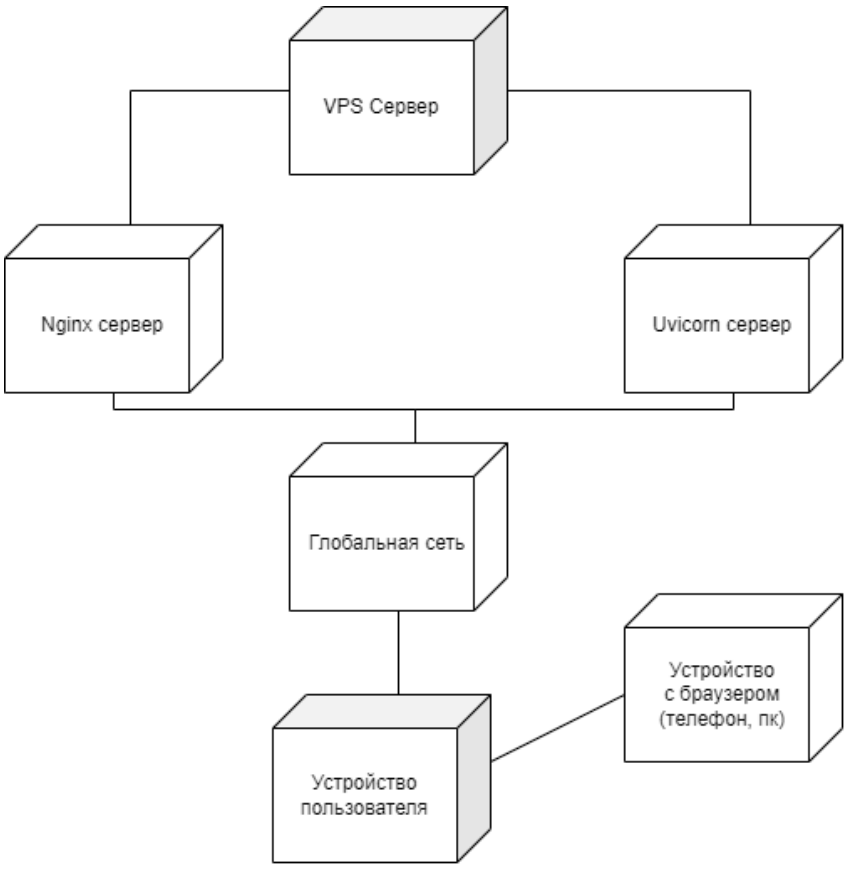


Рисунок 17 - Диаграмма развертывания разрабатываемого приложения

2.2 Разработка модели предметной области

Пользуясь списком категорий и методом анализа словесного описания вариантов использования, составлен список кандидатур на роль концептуальных классов для предметной области. Он соответствует требованиям и принятым упрощениям для всей предметной области. Список концептуальных классов:

— пользователи;

— комнаты;

— учебные материалы;

— домашние работы;

— выполненные домашние работы;

— участия;

— сообщения;

— вложения;

На основании анализа словесного описания варианта использования, составлен список ассоциаций для предметной области, представленный в таблице 6.

Таблица 6 – Ассоциации для модели предметной область

Ассоциация	Описание ассоциации
Отправлено	Сообщение отправлено пользователем
Получено	Сообщение получено пользователем
Участвует	Пользователь участвует в процессе обучения в комнате
Приложено	Файлы (вложения) приложены к материалу
Создано	Учебный материал, вложения или домашние работы созданы пользователем
Выложено в	Учебные материалы или домашние работы выложены в комнате
Выполнено	Домашнее задание выполнено для соответствующего задания

На основании анализа технического задания и описания вариантов использования выделены атрибуты классов для модели предметной области, представленные в таблице 7.

В результате объединения концептуальных классов, ассоциаций и атрибутов классов концептуальная модель предметной области имеет вид, показанный на рисунке 8.

Таблица 7 – Атрибуты классов для модели предметной области

Название класса	Атрибуты класса
Пользователь	Id, имя, фамилия, отчество, номер телефона, адрес электронной почты, полученные сообщение, отправленные сообщения
Сообщение	Id, отправитель, получатель, текст, дата отправки
Учебный материал	Id, название, текст, описание, вложения
Вложение	Id, байтовые данные, название файла,
Домашнее задание	Id, название, текст, описание, вложения
Выполненное домашнее задание	Id, домашнее задание, оценка, статус проверки
Комната	Id, дата создания, создатель, название, описание

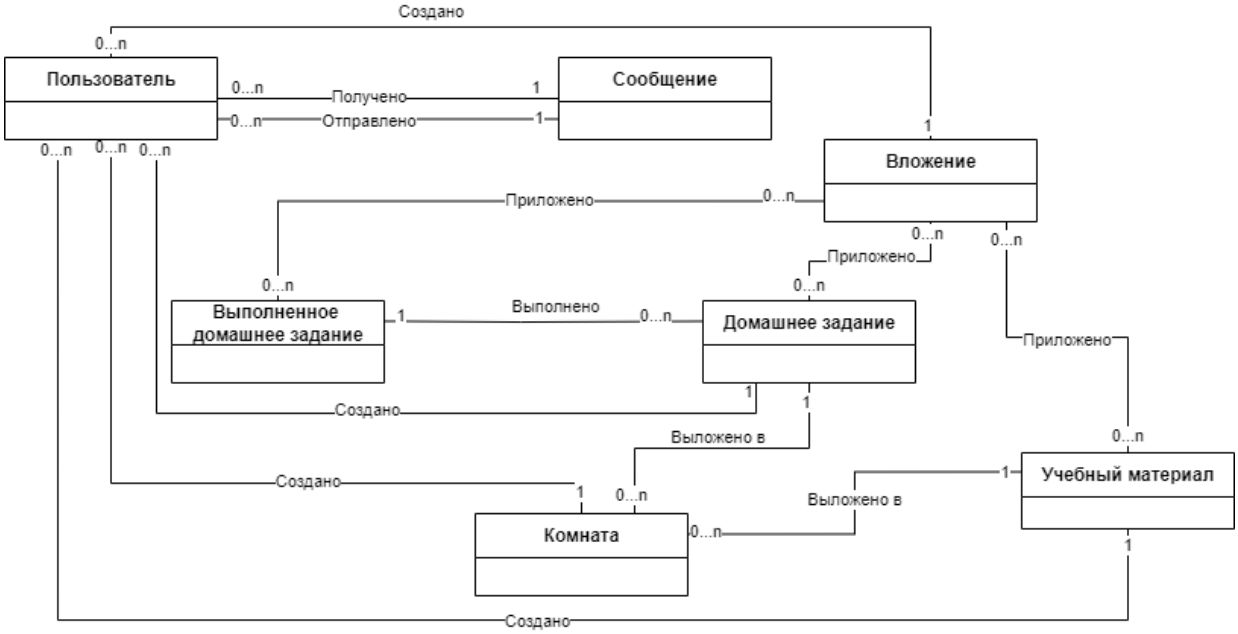


Рисунок 18 – Концептуальная модель предметной области

2.3 Разработка алгоритма функционирования системы

Для того, чтобы распознавать роли пользователя в комнате и его права необходимо будет авторизоваться в системе. Набор возможностей будет отличаться для пользователя, имеющего статус ученика, либо же преподавателя в какой-то конкретной комнате.

Для этого необходимо разработать разные интерфейсы для работы преподавателя или ученика.

Алгоритм работы ученика и преподавателя в виде диаграммы деятельностей приведен на рисунке 19.

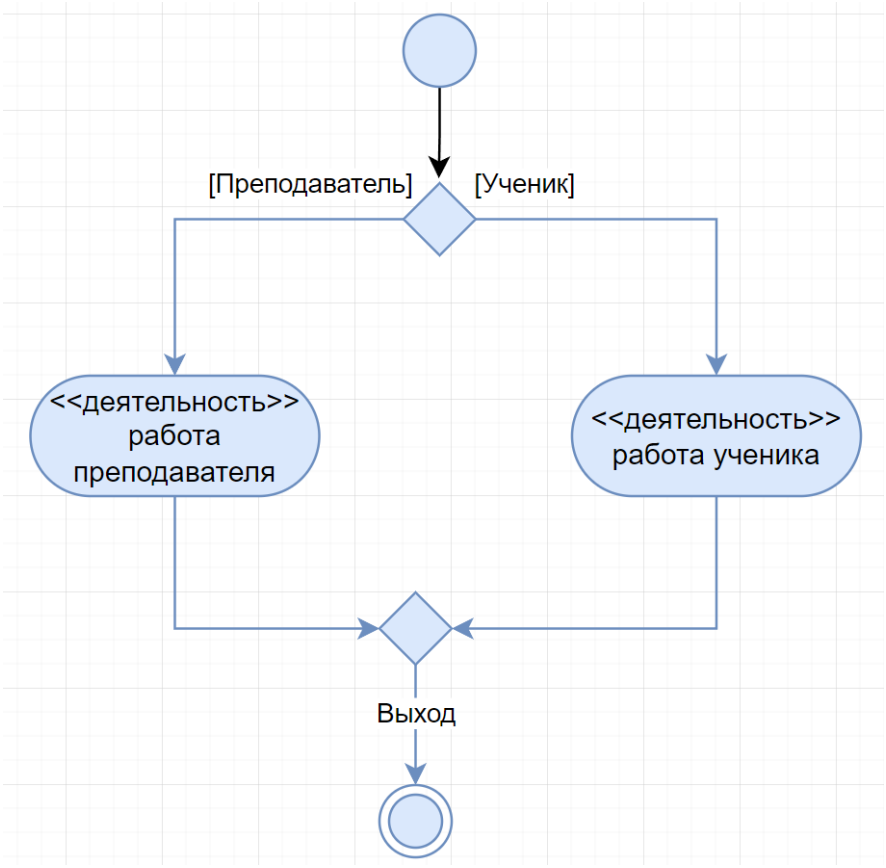


Рисунок 19 - Алгоритм работы ученика и преподавателя в виде диаграммы деятельностей

Алгоритм работы преподавателя представлен на рисунке 20.

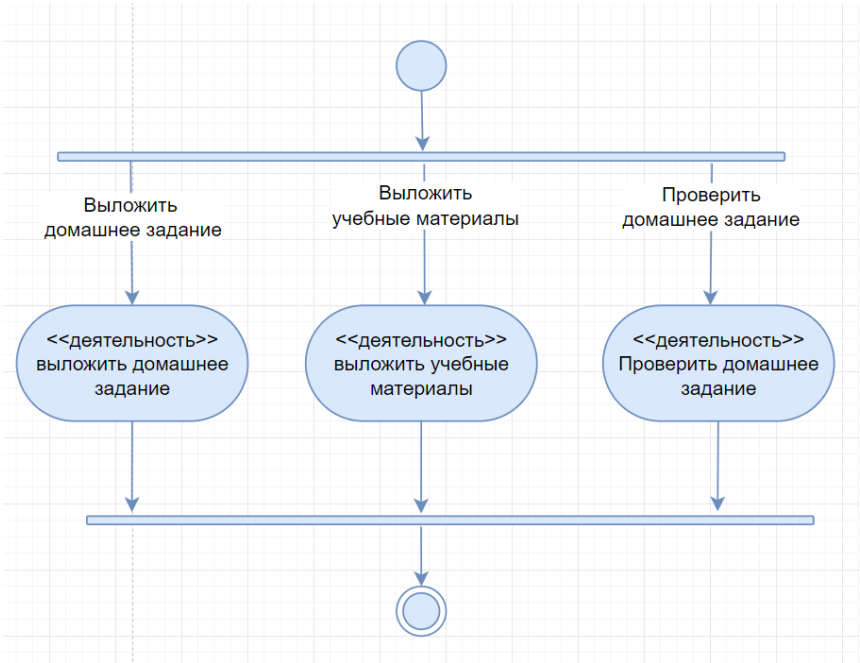


Рисунок 20 – Алгоритм работы преподавателя

На рисунке 21 представлен алгоритм деятельности преподавателя «выложить учебные материалы».

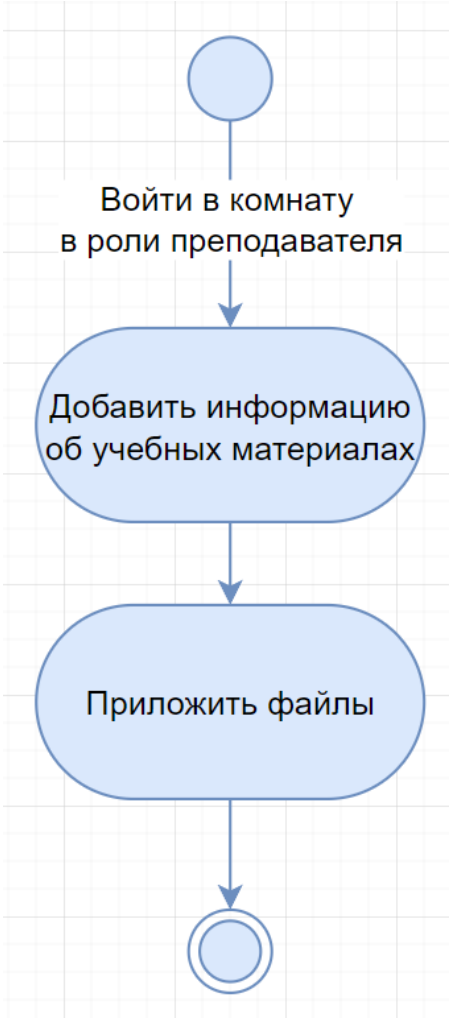


Рисунок 21 - Алгоритм деятельности преподавателя «выложить учебные материалы».

На рисунке 22 представлен алгоритм деятельности преподавателя «выложить домашнее задание».

Рисунок 22 - Алгоритм деятельности преподавателя «выложить домашнее задание».

На рисунке 23 представлен алгоритм деятельности преподавателя «проверить домашние задания».

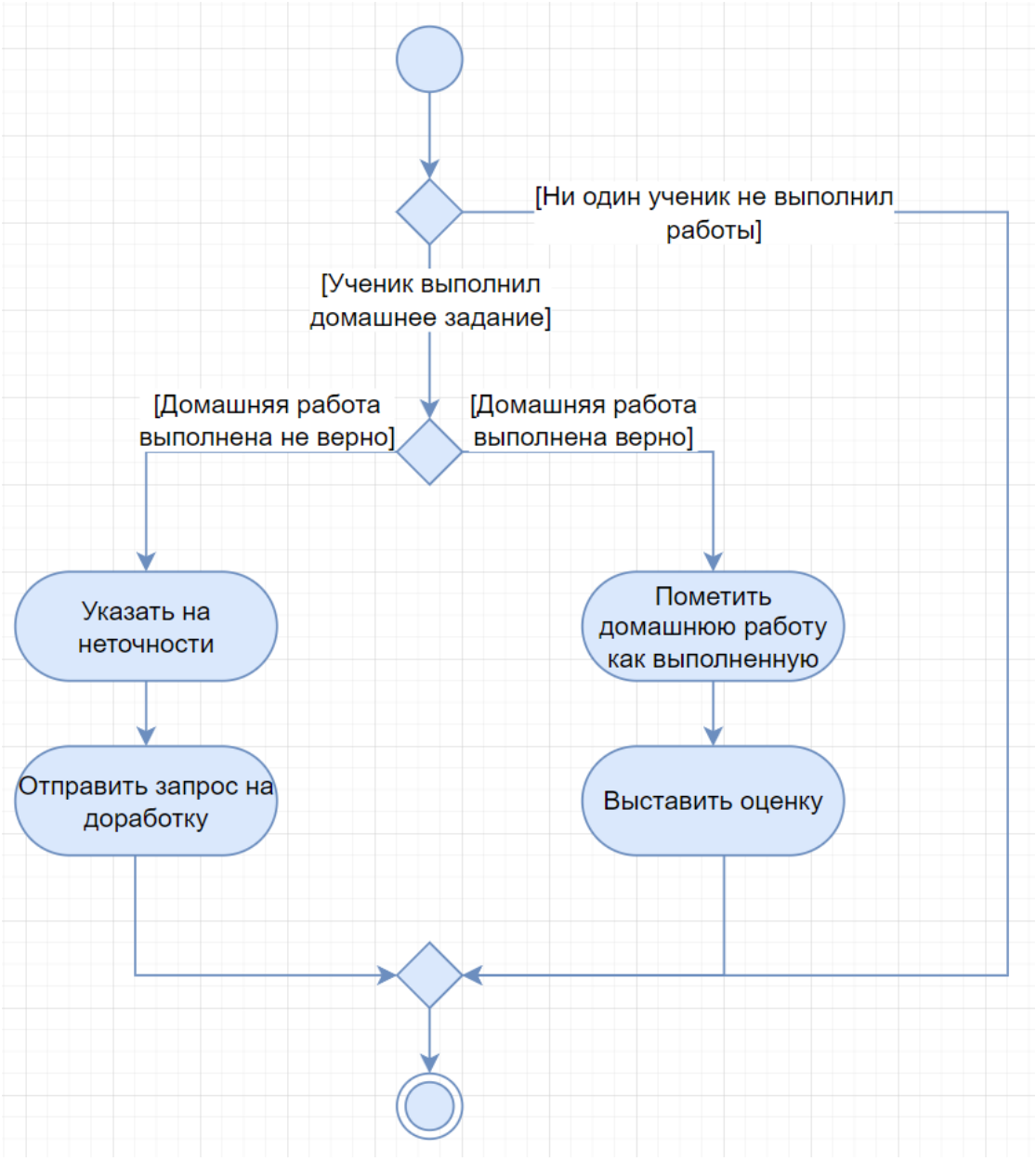


Рисунок 23 - Алгоритм деятельности преподавателя «проверить домашние задания».

На рисунке 24 представлен алгоритм деятельности «работа ученика».

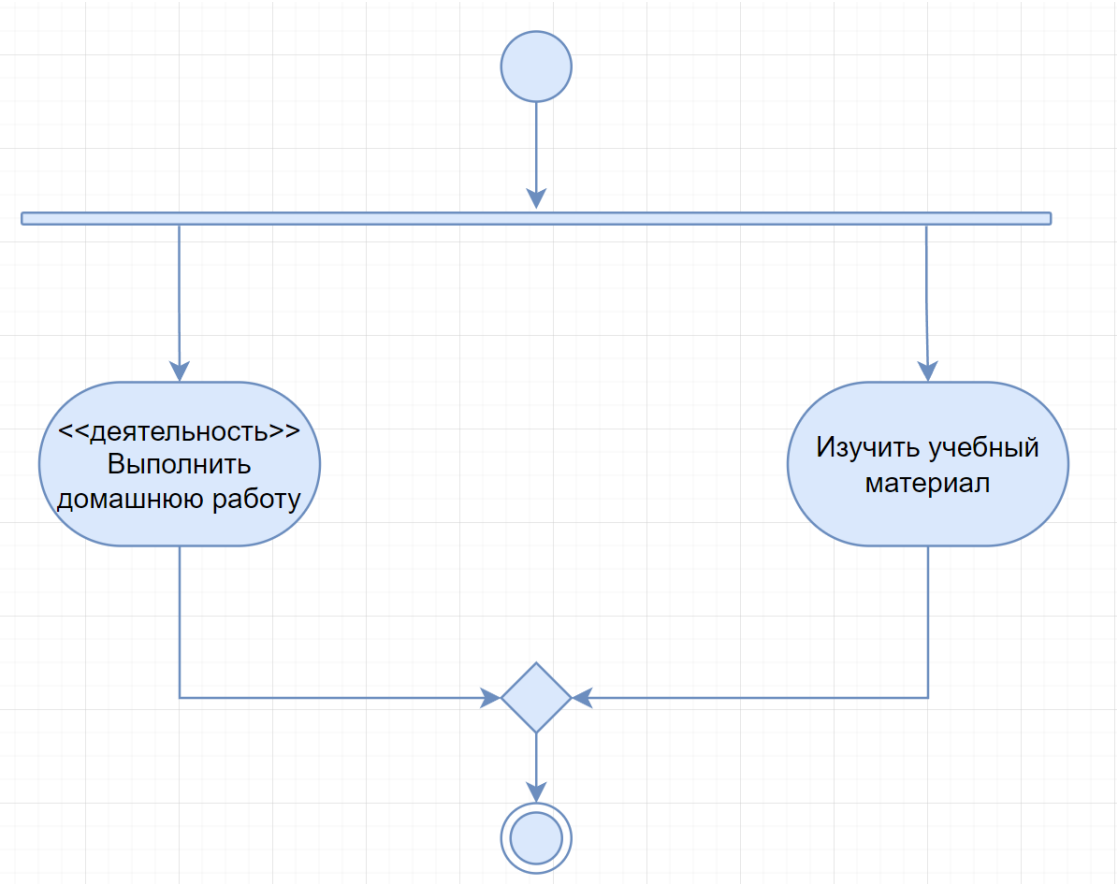


Рисунок 24 - Алгоритм деятельности «работа ученика».

На рисунке 25 представлен алгоритм деятельности «выполнить домашнюю работу».

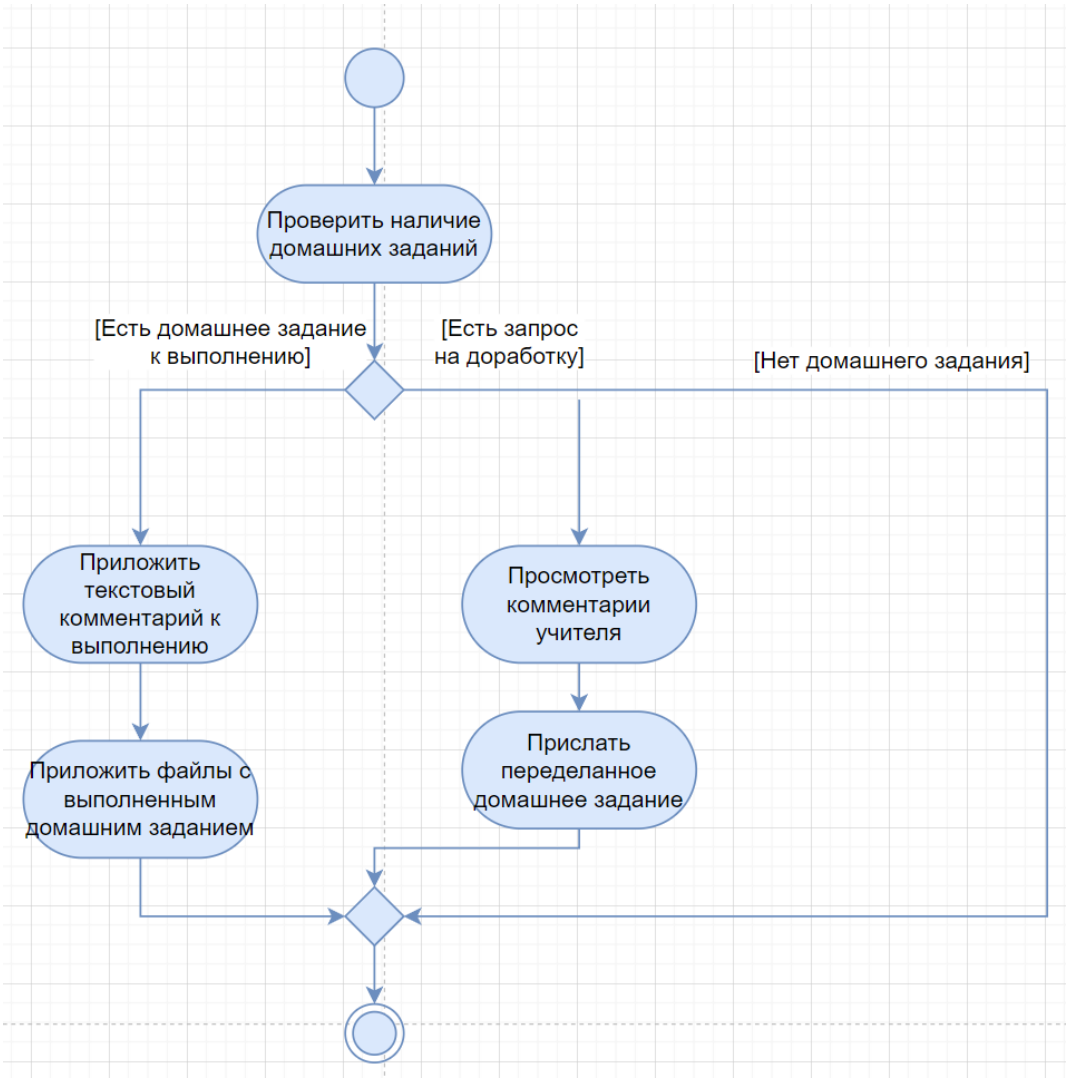


Рисунок 25 - Алгоритм деятельности ученика «выполнить домашнюю работу».

2.4 Проектирование интерфейса пользователя

На основании алгоритма функционирования и требований к интерфейсу разработана диаграмма состояний, представленная на рисунке 26.

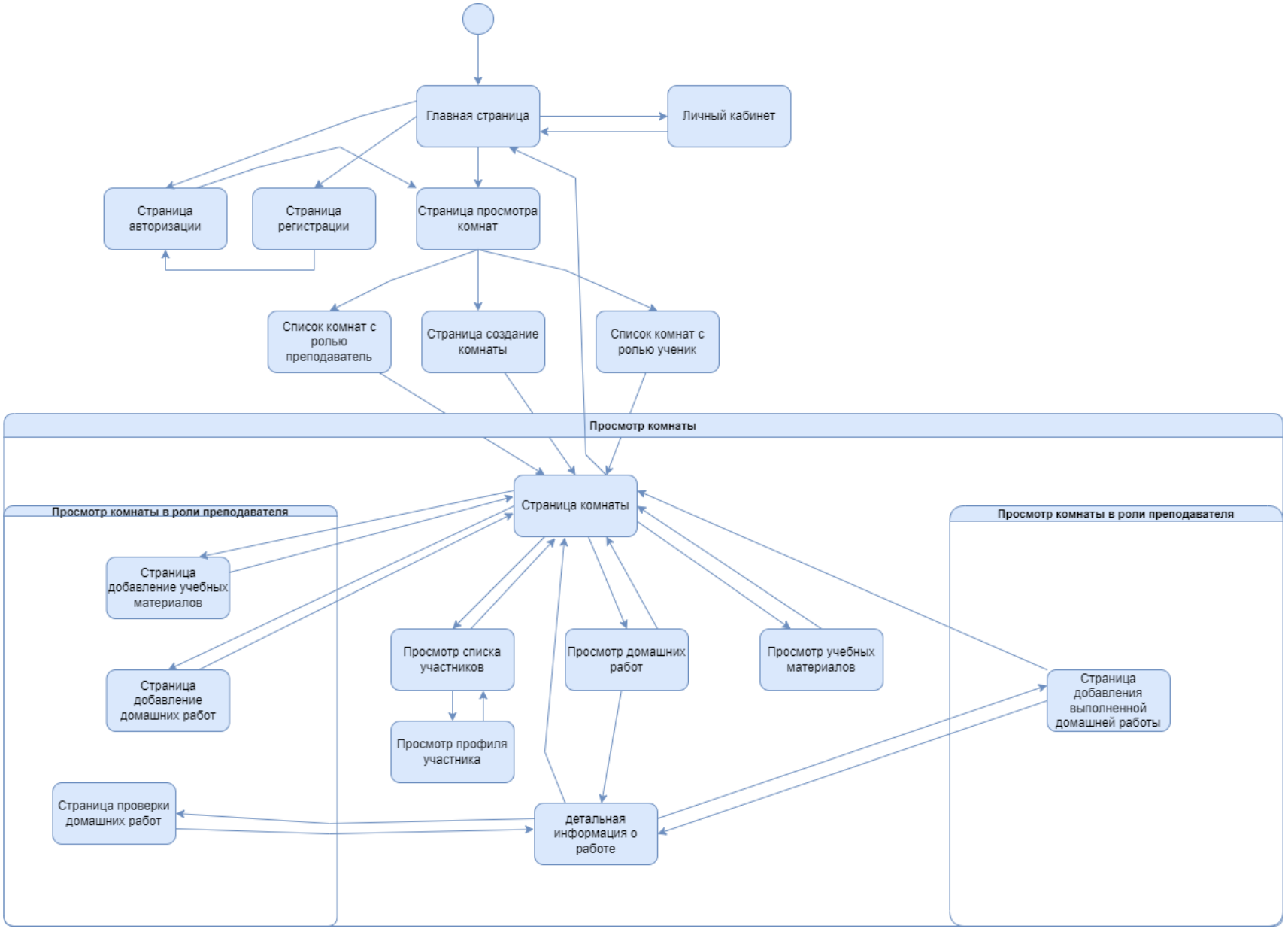


Рисунок 26 - Диаграмма состояний интерфейса пользователя

После перехода пользователя на сайт в браузере появляется ДОПИСАТЬ ПОСЛЕ ГОТОВОГО ИНТЕРФЕЙСА

2.5 Реляционная модель данных

На рисунке 27 изображена реляционная модель данных.

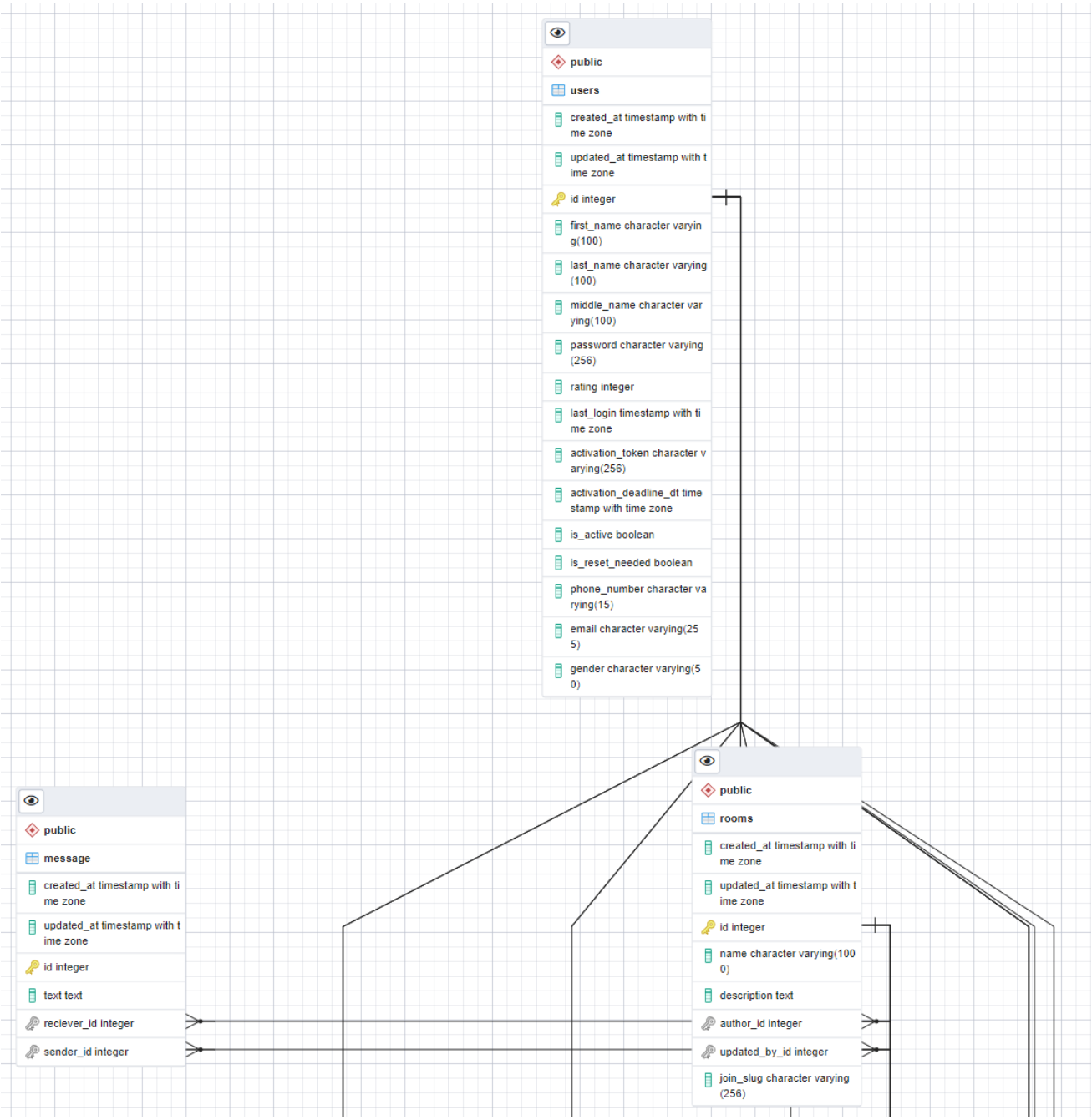


Рисунок 27 - Реляционная модель данных

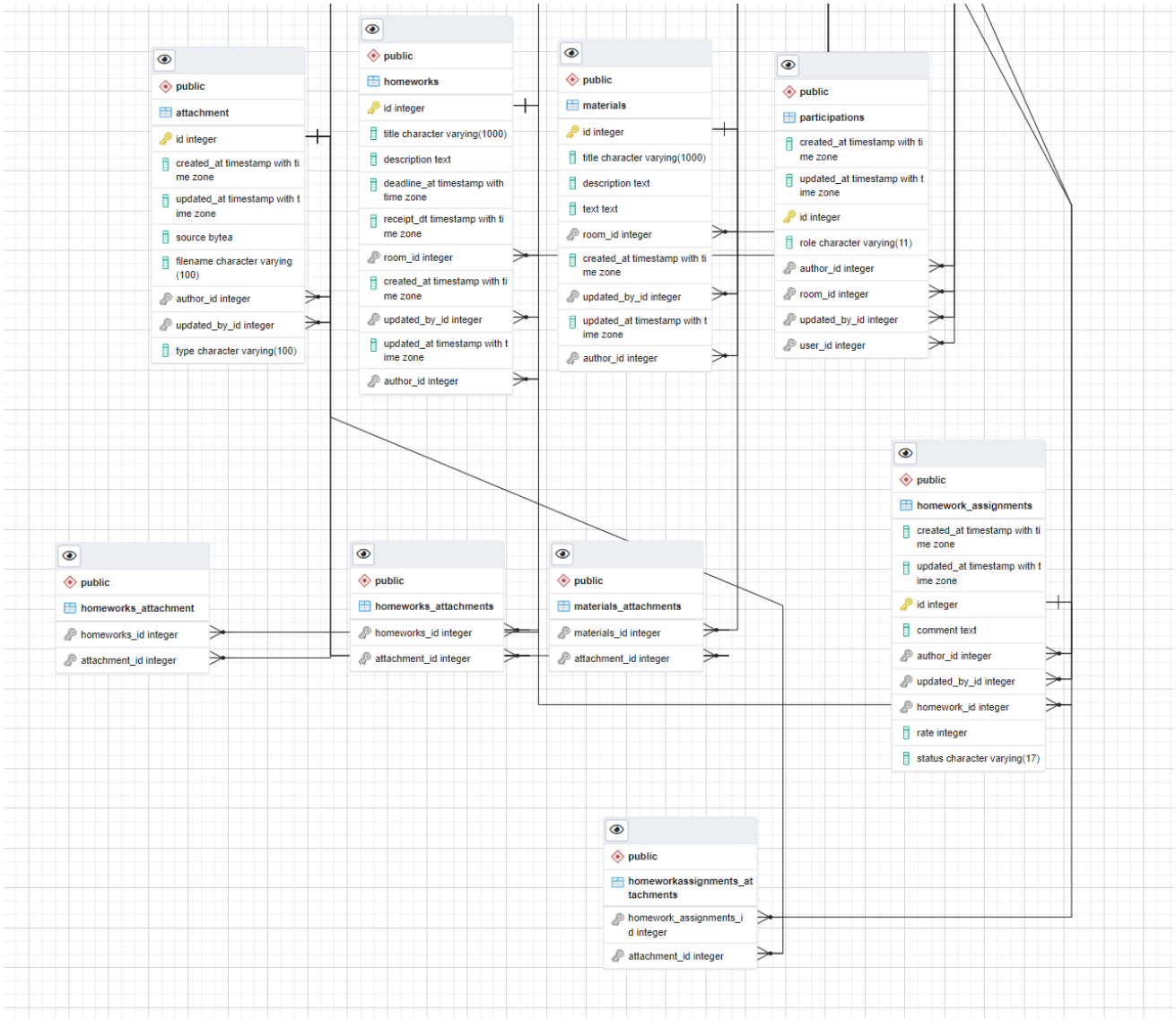


Рисунок 27 (продолжение) - Реляционная модель данных

Реляционная модель данных разработана на основе концептуальной модели предметной области. Как видно на диаграмме, довольно большое количество сущностей имеют отношение 1:M и M:M. 1:M реализуется за счёт добавления внешнего ключа в таблицу со степенью связи M. Связь M:M реализуется за счёт связующей таблицы, содержащей в себе первичные ключи обоих сущностей, ссылающихся друг на друга. Данная разработанная реляционная модель в дальнейшем послужит для разработки самой базы данных.

2.6. Проектирование классов предметной области

2.6.1. Построение диаграмм последовательностей для варианта использования «Построить таблицу событий»

Диаграмма последовательности, описывающая основной поток событий и его подчиненные потоки изображена на рисунке 28.

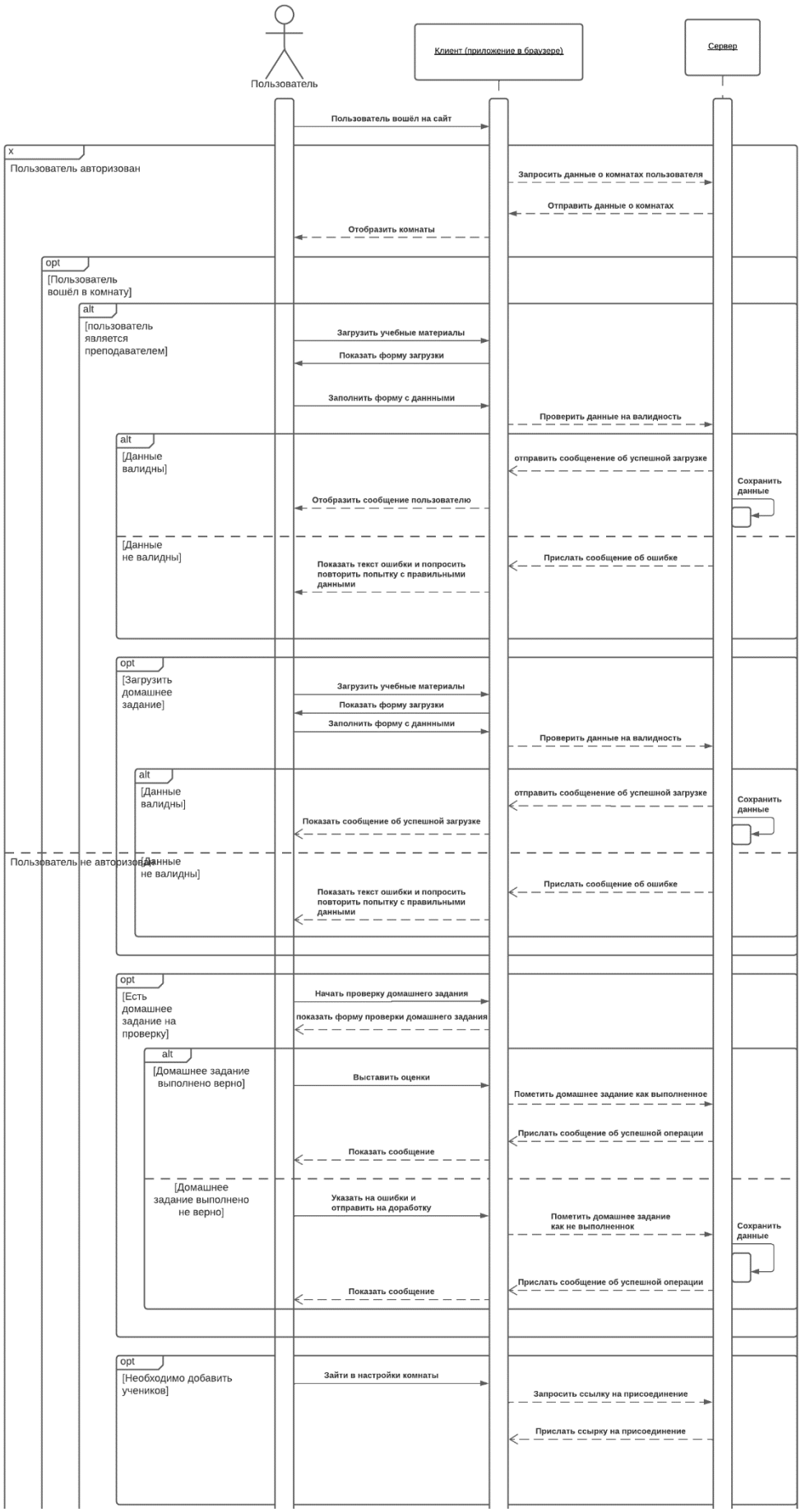


Рисунок 28 - Диаграмма последовательности, описывающая основной поток (часть 1)

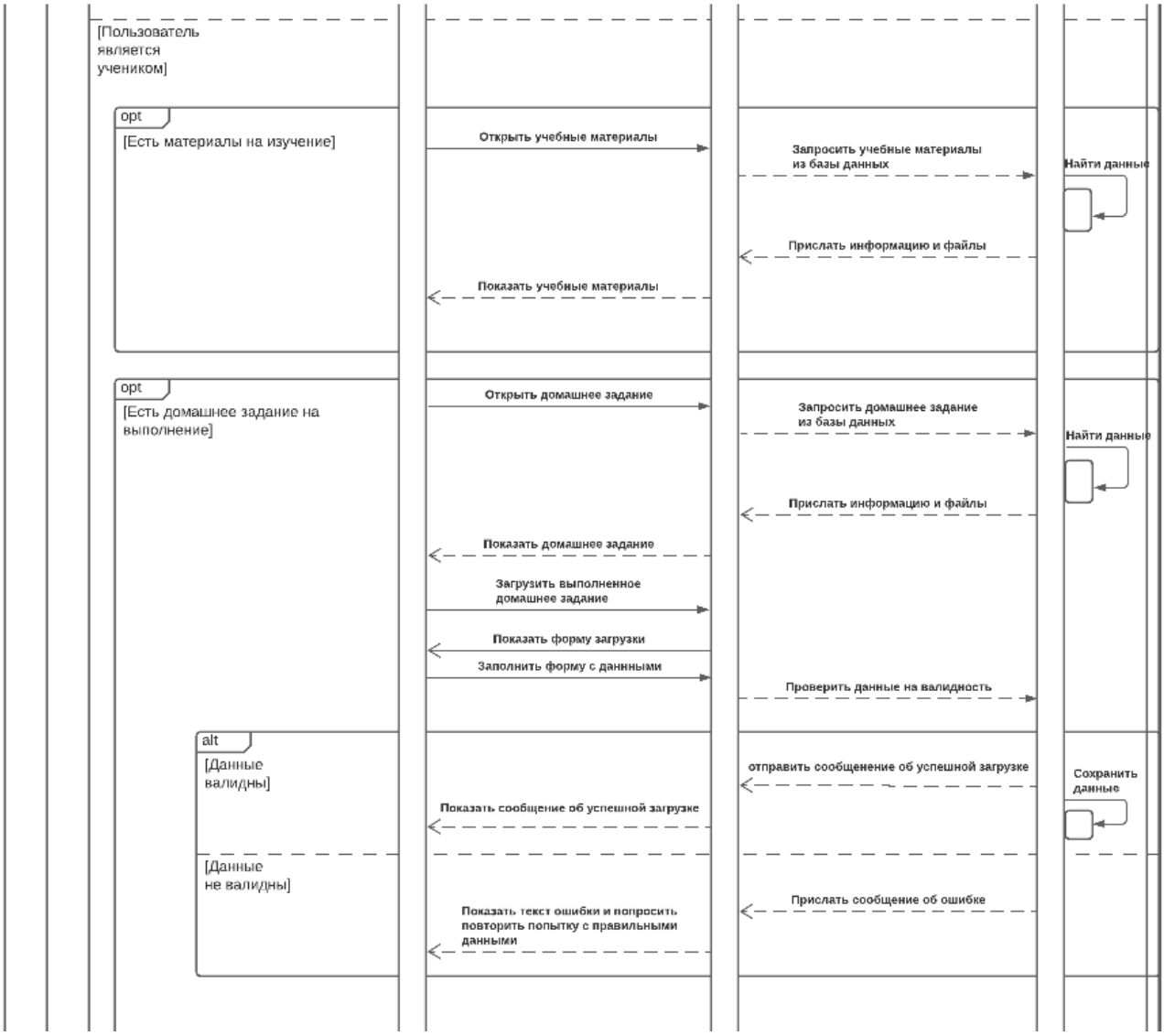


Рисунок 28 - Диаграмма последовательности, описывающая основной поток (часть 2)

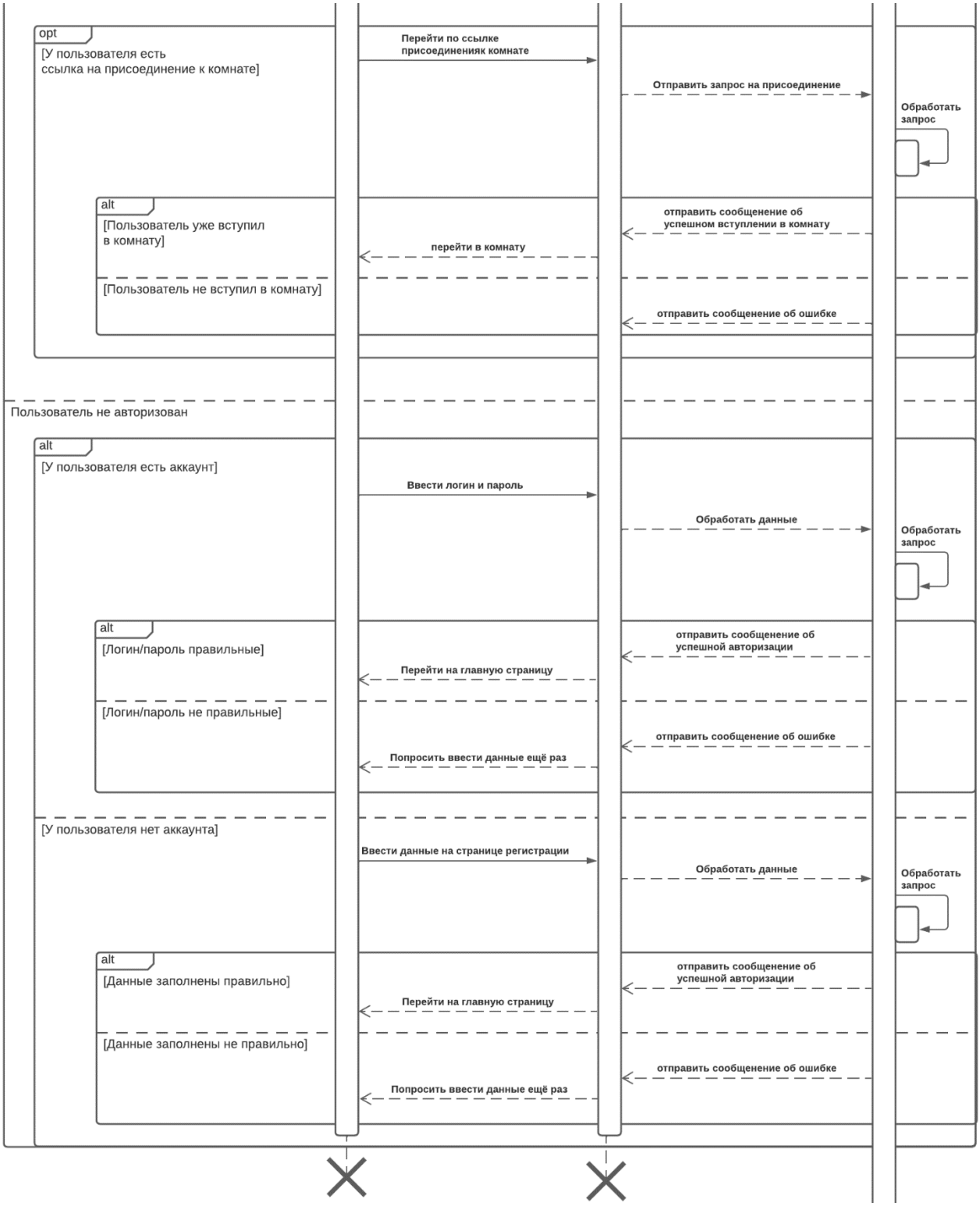


Рисунок 28 - Диаграмма последовательности, описывающая основной поток (часть 3, конец)

2.6.2 Построение диаграммы кооперации

Структурные особенности передачи и приема сообщений между объектами представлены на диаграмме кооперации на рисунке 29.

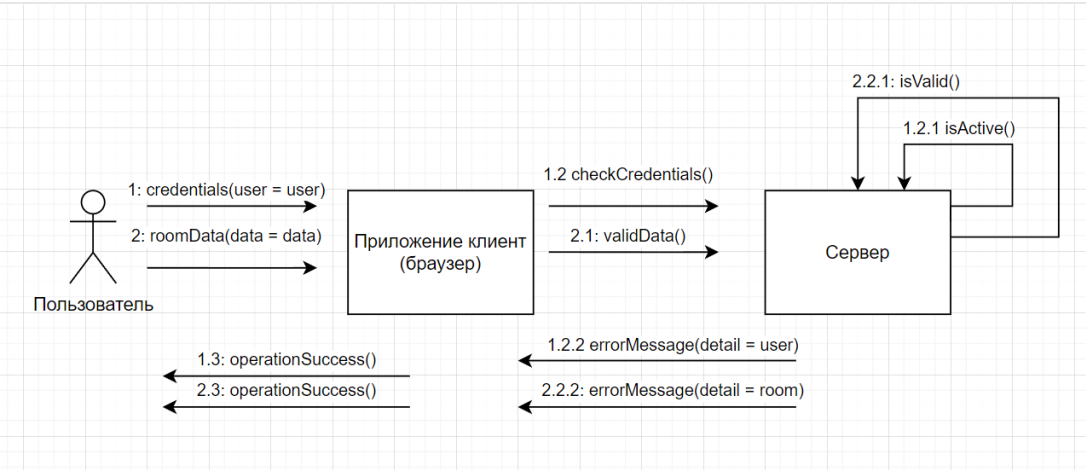


Рисунок 29 – Диаграмма кооперации

2.6.3 Построение диаграммы классов

Диаграмма классов представлена на рисунке 30.

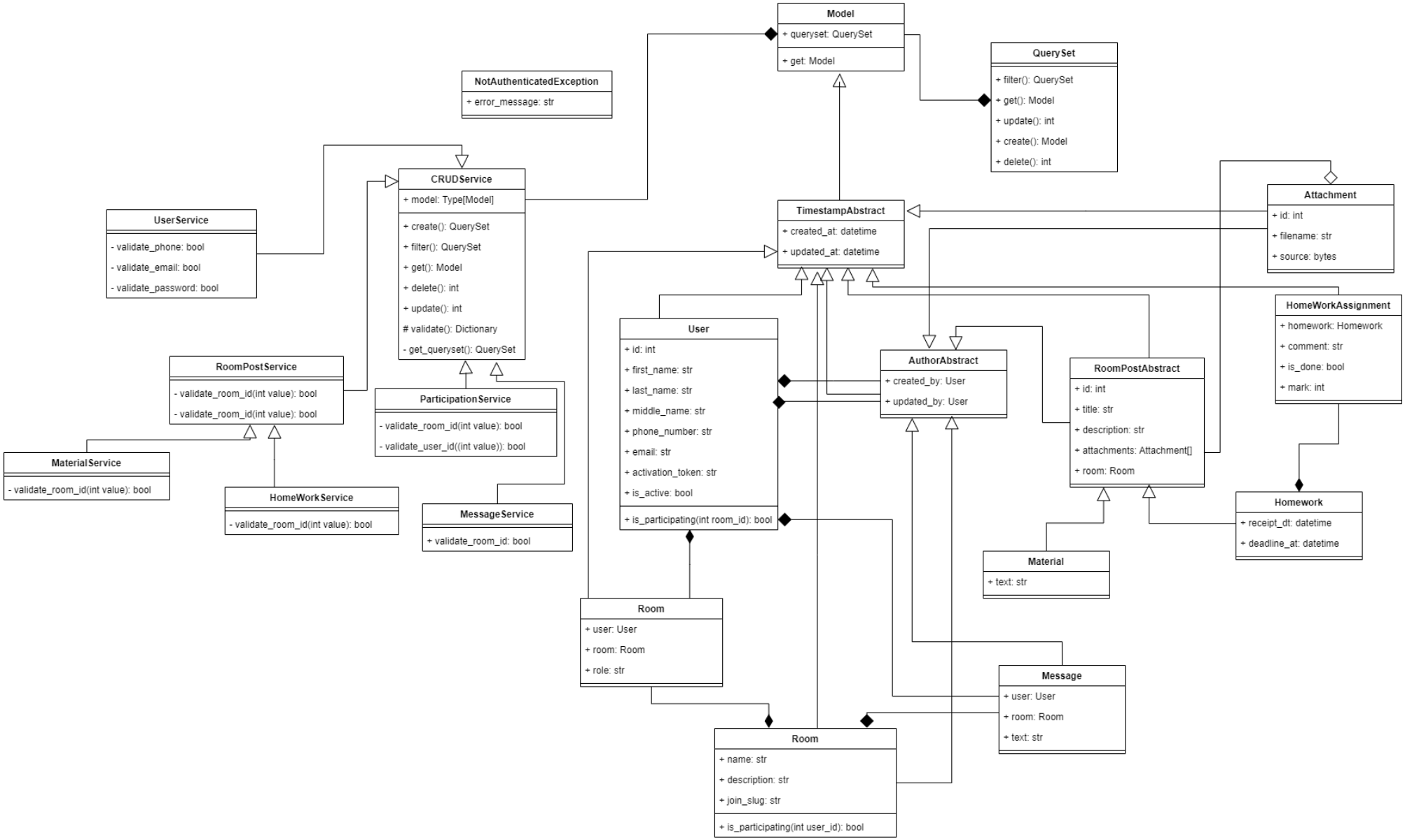


Рисунок 30 – Диаграмма классов

Разрабатываемая архитектура приложения проектировалась с учётом современного подхода Domain Driven Design, которая подразумевает под собой разделение приложения на отдельные слои для работы с базой данных, данными пользователя и внешними данными, приходящими из запросов к серверу.

Кроме того, как видно по диаграмме, само приложении написано в виде микросервисной архитектуры, имея свои сервисы для работы с каждым видом данных (например, для работы с комнатами используется класс RoomService). Кроме того, очень многие модели наследуются от класса TimeStampAbstract и класса AuthorAbstract, которые хранят в себе информацию о дате загрузи какого-либо контента или создании комнаты, и их авторе. Это сделано, с целью удобного отслеживания информации о добавлении и изменении объектов в системе.

2.6.4 Уточнение структуры классов предметной области и разработка алгоритмов методов

Класс CRUDService отвечает за основные операции, связанные с базой данных а конкретно операции чтения, записи, удаления и обновления данных. Данный класс является абстрактным и зависит в основном от переданной в него модели. Например, если у наследника класса CRUDService определить свойство model как Room, то, соответственно все основные операции будут проводится над таблицей «Комната». Метод validate описан таким образом, что во время его вызова будут вызваны все методы вида: validate_<имя поля модели>, чтобы произвести валидацию соответствующего поля. Если валидация прошла успешна, то метод возвращает true, иначе – false. На этом методе завязана работа многих его классов наследников.

Таблица 8 – Атрибуты класса CRUDService

Атрибут	Тип	Описание
model	Type[Model]	Тип данных, над которыми будут производится операции

Класс User представляет из себя таблицу в базе данных «users» выполненную в виде класса, для удобной работы с записями таблицы в виде объектов в самом языке программирования.

Таблица 9 – Атрибуты класса User

Атрибут	Тип	Описание
id	int	Первичный ключ
first_name	string	Имя
last_name	string	Фамилия
middle_name	string	Отчество
phone_number	string	Номер телефона
email	string	Адрес электронной почты
activation_token	string	Код активации
is_active	bool	Признак
created_at	datetime	Дата создания пользователя
updated_at	datetime	Дата последнего редактирования

Таблица 10 – Атрибуты класса Room

Атрибут	Тип	Описание
id	int	Первичный ключ
name	string	Название
description	string	Описание
join_slug	string	Токен для присоединения

Таблица 11 – Атрибуты класса Participation

Атрибут	Тип	Описание
id	int	Первичный ключ
role	string	Текстовое значение роли (преподаватель или ученик)
user	User	Вступивший пользователь
room	Room	Комната

Таблица 12 – Атрибуты класса Material

Атрибут	Тип	Описание
id	int	Первичный ключ
title	string	Название
description	string	Краткое описание
text	string	Текстовое пояснение
author	User	Вступивший пользователь
room	Room	Комната
attachments	Attachment[]	Массив вложенных файлов

Таблица 13 – Атрибуты класса Homework

Атрибут	Тип	Описание
id	int	Первичный ключ
title	string	Название
description	string	Краткое описание
author	User	Вступивший пользователь
room	Room	Комната
attachments	Attachment[]	Массив вложенных файлов

Таблица 14 – Атрибуты класса Homework

Атрибут	Тип	Описание
id	int	Первичный ключ
title	string	Название
description	string	Краткое описание
author	User	Вступивший пользователь
room	Room	Комната
attachments	Attachment[]	Массив вложенных файлов

Таблица 15 – Атрибуты класса HomeworkAssignment

Атрибут	Тип	Описание
id	int	Первичный ключ
title	string	Название
comment	string	Краткое описание
status	string	Статус сданной работы
author	User	Вступивший пользователь
homework	Homework	Домашняя работа
attachments	Attachment[]	Массив вложенных файлов

Таблица 16 – Атрибуты класса Attachment

Атрибут	Тип	Описание
id	int	Первичный ключ
source	bytes	Байты с исходным файлом
filename	string	Название файла
type	string	Тип файла

3 Реализация системы

3.1 Реализация программного обеспечения системы

3.1.1 Разработка диаграммы компонентов

Поскольку данное приложение является клиент-серверным, то для его реализации были выбраны разные технологии и два разных языка программирования.

Для реализации серверной части был выбран язык Python с фреймворком FastAPI и субд PostgreSQL. Язык программирования python за последнее время очень неплохо зарекомендовал себя в веб разработке, а большое количество сторонних высокоуровневых библиотек и встроенных синтаксических конструкций (так называемый синтаксический сахар) позволяют в кратчайшее время приступить к разработке самого приложения, не зацикливаясь на реализации однотипных низкоуровневых алгоритмов. Сам же фреймворк FastAPI является одной из самых новых и за последнее время всё чаще применяемых open-source технологий, обеспечивающий быстрый старт для разработки RESTAPI и автоматической генерации схемы OpenAPI, что по сути является автоматической документацией к написанному приложению. СУБД PostgreSQL неплохо зарекомендовала себя в удобстве как для администраторов баз данных, так и для разработчика. Она очень легко разворачивается, хорошо выдерживает нагрузки (если программист пишущий SQL запросы не пытается их нарочно усложнить) и очень удобна в использовании разработчику благодаря наличию удобного средства администрирования - PG Admin.

Для реализации клиентской части был выбран фреймворк JavaScript с фреймворком Vue.js. Vue.js позволяет сделать быстрое, легковесное клиентское приложение с динамической генерацией Web страничек в зависимости от пришедшего с сервера ответа от RESTAPI. Для работы с сервером была использована встроенная библиотек fetch api.

Реализация программного обеспечения системы представлена на рисунке 31 в виде диаграммы компонентов. Она определяет архитектуру разрабатываемой системы на физическом уровне и представляет зависимости между программными компонентами.

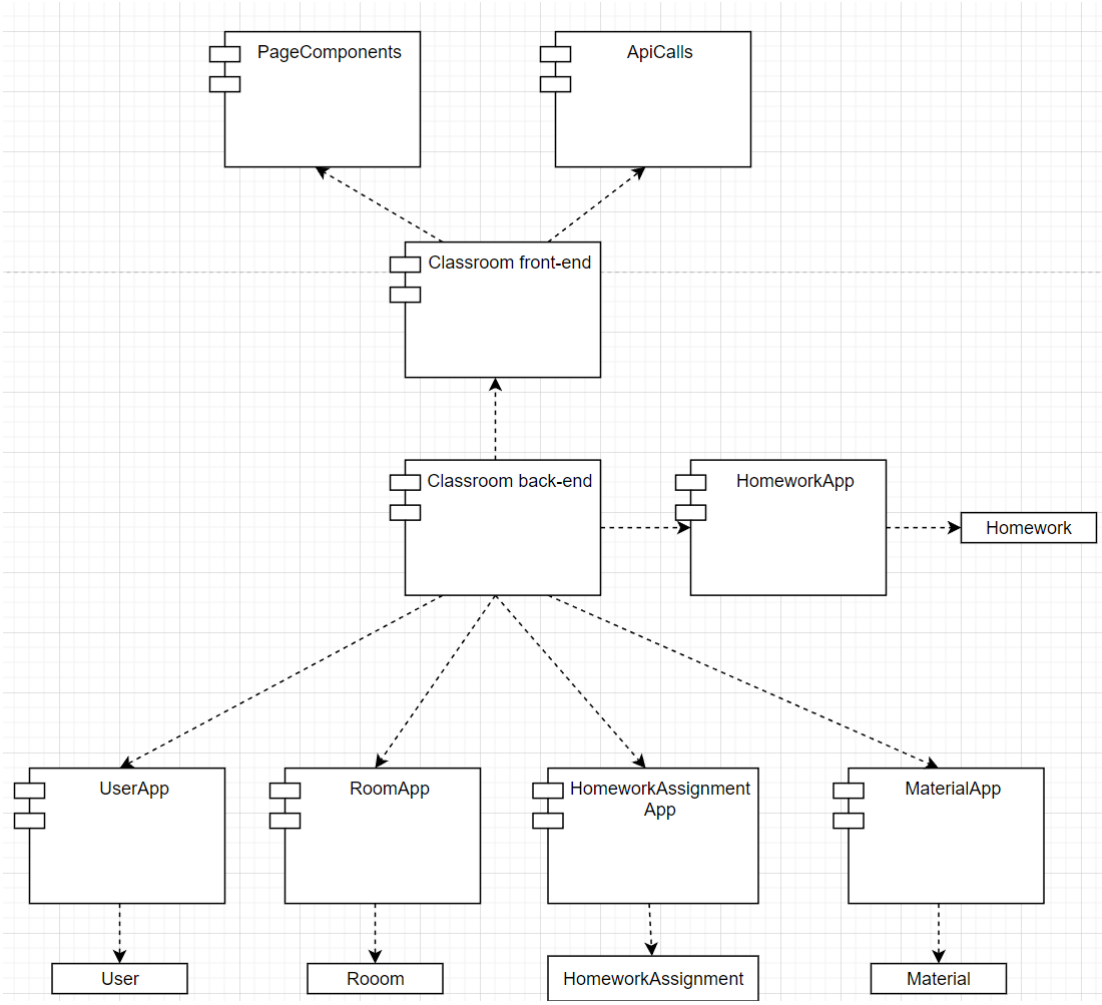


Рисунок 31 – Диаграмма компонентов приложения

3.1.2 Объекты интерфейса пользователя

Программный продукт состоит из нескольких совместно работающих сервисов: UserApp, RoomApp, HomeworkAssignmentApp, MaterialApp, HomeworkApp, каждый из которых отвечает за реализацию какой-то конкретной программной сущности. Например, UserApp – сервис, отвечающий за авторизацию и работу с данными пользователя, RoomApp – сервис, отвечающий за работу пользователя с комнатами, HomeworkApp – сервис, отвечающий за работу пользователя с домашними работами, HomeworkAssignmentApp – сервис, отвечающий за работу с выполненными домашними заданиями. Со стороны клиента они представляют собой несколько секция разбитых на вкладки Profile, Education и Main - общая информация о странице, доступная всем.

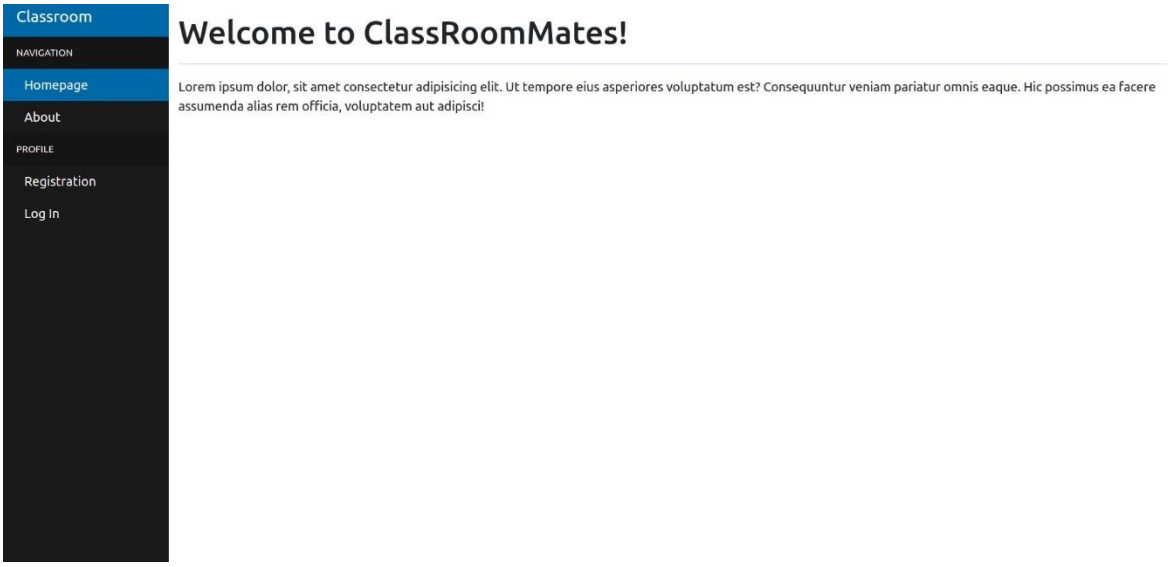


Рисунок 32 – Макет интерфейса главной страницы

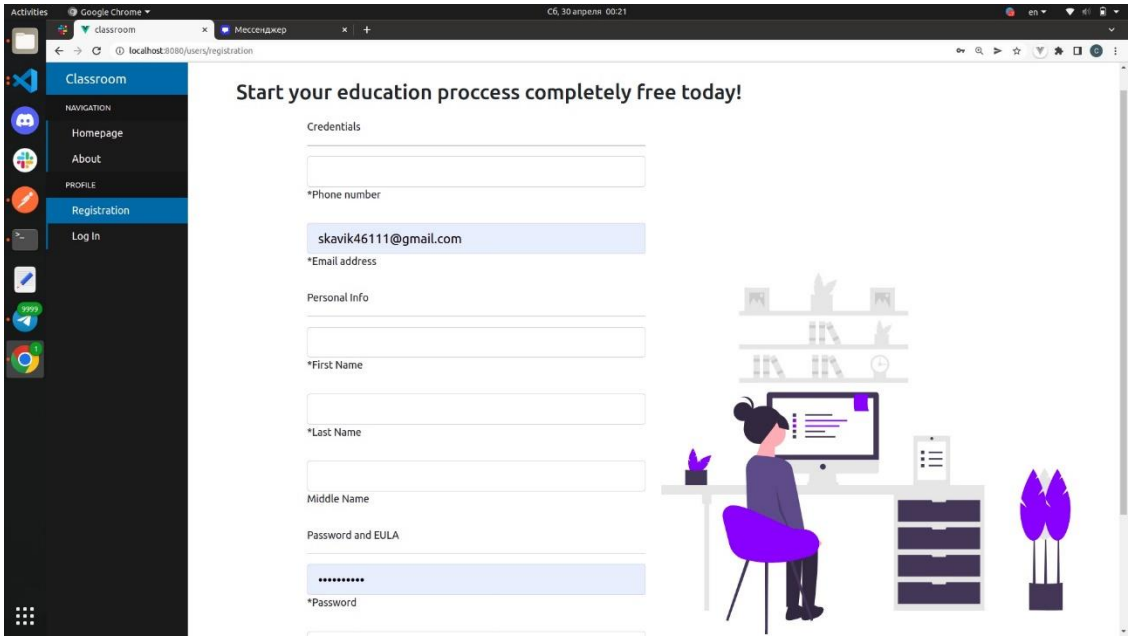


Рисунок 33 – Макет интерфейса страницы регистрации

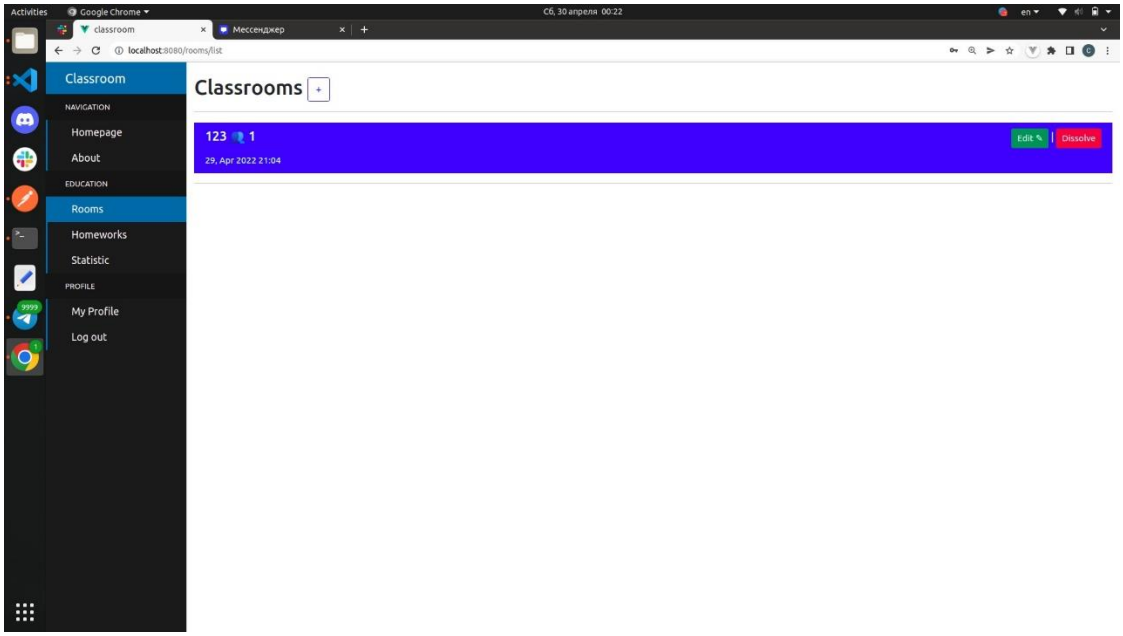


Рисунок 34 – Макет интерфейса страницы с комнатами

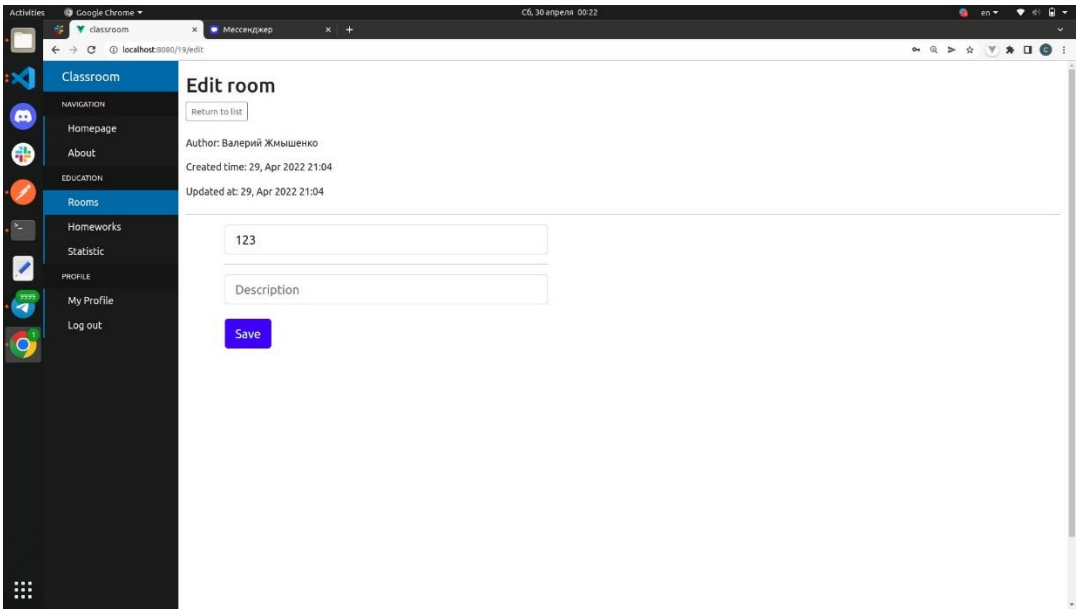


Рисунок 35 – Макет интерфейса создания комнаты

В таблице 2 представлено описание элементов управления.

Таблица 2 – Описание макета графического интерфейса пользователя

Название поля	Тип	Условия видимости	Условия доступности	Описание
Save	Кнопка	Виден всегда	Доступен авторизованным пользователям	Кнопка сохранения данных
LogIn	Кнопка	Виден не зарегестрированным пользователям	Доступна после заполнения полей	Кнопка логина
Classroom	Панель навигации	Виден всегда	Доступен на любой странице	Панель
Welcome to classroom mates!	Надпись	Видна на главной странице	Доступна всем	Заголовок главной страницы с приветствием
Start your education process completely free today!	Надпись	Видна на странице регистрации	Доступна всем	Надпись при регистрации
Name	Текстовое поле	Виден на странице изменения и создания комнаты	Доступна всем	Поле для ввода названия комнаты
Description	Текстовое поле	Виден на странице изменения и создания комнаты	Доступна всем	Поле ввода описания комнаты

Продолжение таблицы 2

Название поля	Тип	Условия видимости	Условия доступности	Описание
Phone	Текстовое поле	Виден на странице авторизации и регистрации	Доступен всегда	Поле для ввода номера телефона
Email	Текстовое поле	Виден на странице авторизации и регистрации	Доступен всегда	Поле для ввода email
First Name	Текстовое поле	Виден на странице регистрации	Доступен при регистрации	Поле для ввода имени
Last Name	Текстовое поле	Виден на странице регистрации	Доступен при регистрации	Поле для ввода фамилии
Middle Name	Текстовое поле	Виден на странице регистрации	Доступен при регистрации	Поле для ввода отчества
Return to list	Кнопка	Видна во вкладке Rooms	Доступен при создании и редактировании комнаты	Кнопка для возвращения в список комнат

3.2 Реализация технического обеспечения

Полная диаграмма развертывания системы обмена файлами и сообщениями для организации образовательного процесса приведена на рисунке 36.

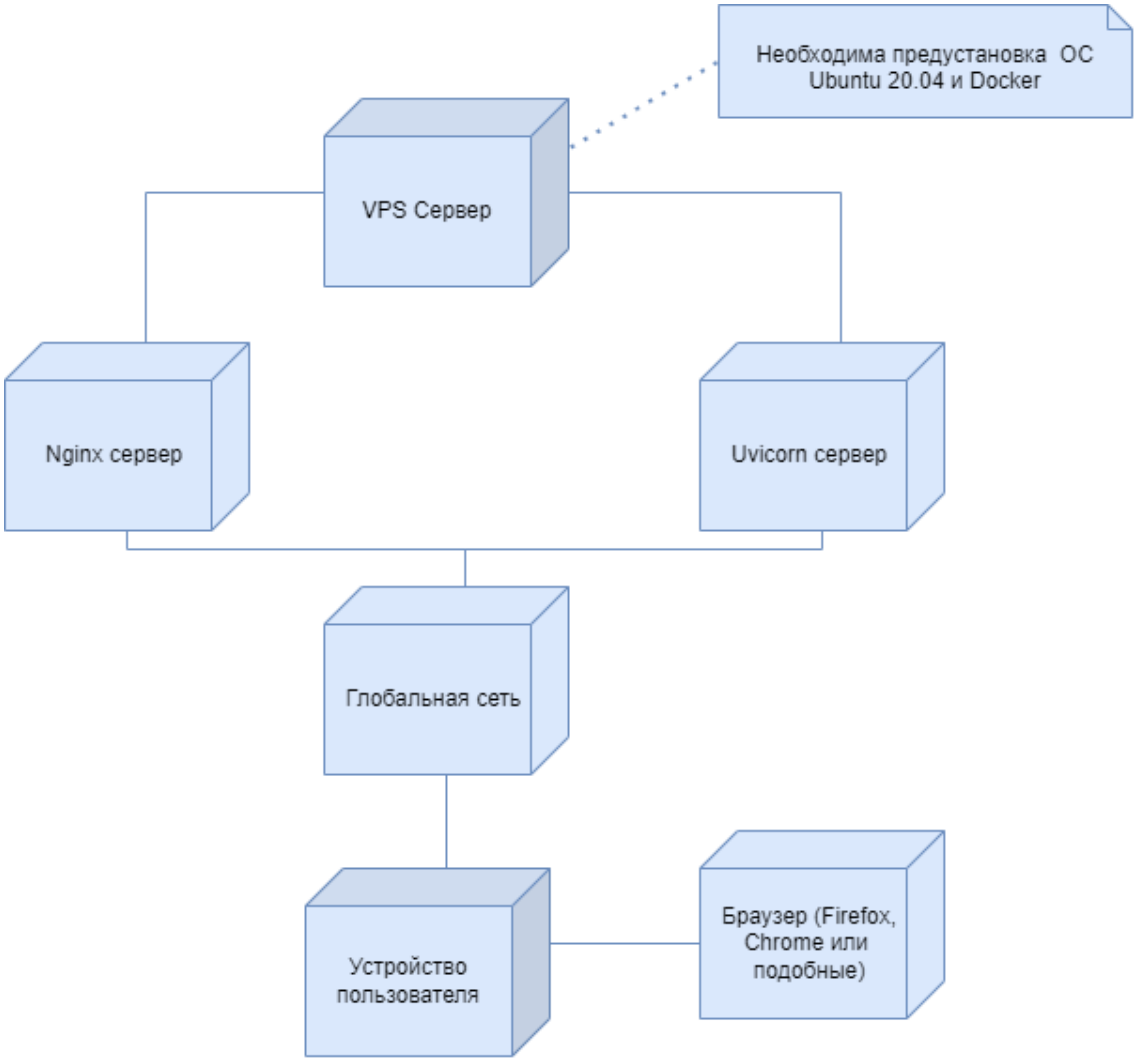


Рисунок 36 – Полная диаграмма развертывания системы обмена файлами и сообщениями для организации образовательного процесса

4 ТЕСТИРОВАНИЕ И ОЦЕНКА ТРУДОЕМКОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

4.1 Тестирование программного продукта

4.1.1 Функциональное тестирование

На основе функциональных требований (1.3.2) могут быть сформулированы следующие тест-требования.

Тест-требования

1. Проверить, что разрабатываемый программный продукт различает роли преподавателя и ученика.

2. Проверить, что разрабатываемый программный продукт обеспечивает комфортный образовательный процесс для студентов и преподавателей.

3. Проверить, что данный продукт обеспечивает обмен файлами между преподавателями и учениками.

4. Проверить, что разрабатываемый программный продукт обеспечивает обмен файлами между преподавателями и учениками.

5. Проверить, что разрабатываемый программный продукт обеспечивает обмен сообщениями между преподавателями и учениками.

6. Проверить, что данный продукт помогает в оценке успеваемости учеников.

Результаты функционального тестирования представлены в таблице 4.1.

Таблица 4.1 – Результаты функционального тестирования

Номер тестового примера	Номер соответствующего тестового требования	Сценарий выполнения тестового примера (действие оператора)	Реакция системы	Результат (пройден / не пройден)
1	2	Пользователь зашёл на главную страницу сайта	Появилась приветствующая форма	пройден
2	1	Пользователь перешёл на страницу регистрации	Показывается форма регистрации	пройден
3	1	Пользователь ввёл данные с ошибкой (неверный формат данных, не заполнены поля, или заполнены данные уже существующего пользователя)	Выводится подробная информация об ошибке рядом с каждым полем	пройден
4	1	Пользователь ввёл правильные данные	Выводится сообщение о регистрации, на почту присылается письмо с активацией аккаунта	пройден
5	1	Переход на страницу авторизации	Пользователь видит форму авторизации	пройден

Таблица 4.1 (продолжение) – Результаты функционального тестирования

Номер тестового примера	Номер соответствующего тестового требования	Сценарий выполнения тестового примера (действие оператора)	Реакция системы	Результат (пройден / не пройден)
6	1	Пользователь ввёл правильные данные для входа, но не активировал учётную запись	Выведено сообщение об ошибке	пройден
7	1	Пользователь ввёл неправильные учётные данные	Выведено сообщение об ошибке	пройден
8	1	Пользователь ввёл правильные данные для входа	Произведён вход в учётную запись и переадрессация на страницу профиля	пройден
9	1	Пользователь пытается отредактировать свой профиль, введя неверные данные (валидация аналогична регистрации)	Выводится сообщение об ошибке под каждым полем	пройден
10	2	Пользователь переходит в список комнат	Пользователь видит все свои комнаты, в которых он участвует. Комнаты в которых он преподаватель выделены специальным цветом	пройден
11	2	Пользователь нажимает на кнопку создания комнаты	Пользователю отображается форма создания комнаты	пройден
12	2	Пользователь создаёт комнату	Выводится сообщение об успешном создании, пользователя перебрасывает в список комнат	пройден

Таблица 4.1 (продолжение) – Результаты функционального тестирования

Номер тестового примера	Номер соответствующего тестового требования	Сценарий выполнения тестового примера (действие оператора)	Реакция системы	Результат (пройден / не пройден)
13	2, 3	Пользователь переходит в комнату	Отображается комната со всеми материалами, домашними заданиями и списком участников	пройден
14	2, 3, 4	Пользователь нажимает на карточку с материалом	Пользователю отображается страница детального просмотра материала со всеми вложениями	пройден
15	2, 3, 4	Пользователь нажал на файл, приложенный к учебному материалу	Браузер начал загрузку файла	пройден
16	2, 3, 4	Пользователь нажал на домашнюю работу в списке	Выводится детальная страница домашнего задания со списком файлов	пройден
17	2, 3, 4, 5	Пользователь нажал на загрузку выполненного домашнего задания	Домашняя работа загрузилась и перевелась в статус «ожидает проверки»	пройден
18	2, 3, 4	Пользователь нажал на список выполненных домашних работ	Пользователь попадает на детальный просмотр выполненной домашней работы	пройден
19	4, 5	Пользователь отправляет на доработку домашнее задание или выставляет оценку	Домашнее задание ставится в статус проверенное	пройден

4.1.2 Тестирование пользовательского интерфейса

На основе требований к интерфейсу (1.3.5) могут быть сформулированы следующие тест-требования.

1. Программный продукт содержит форму авторизации и регистрации пользователя.

2. Пользователю доступны комнаты, в которых происходит образовательный процесс.

3. Пользователю отображаются вкладки с учебными материалами и домашними заданиями.

4. Пользователь может просматривать только комнаты, в которые он вступил.

5. Если пользователь преподаватель, то он может создавать новые учебные материалы в комнате или домашние задания.

6. Интерфейс приложения практичен и интуитивно понятен, а также адоптирован под любые виды экранов на разных устройствах.

Результаты тестирования пользовательского интерфейса представлены в таблице 4.2.

Таблица 4.2 – Результаты тестирования пользовательского интерфейса

Номер тестового примера	Номер соответствующего тестового требования	Сценарий выполнения тестового примера (действие оператора)	Реакция системы	Результат (пройден / не пройден)
1	1	Перейти на страницу авторизации, нажав на кнопку LogIn в панели навигации будучи не авторизованным	Отображается форма авторизации	пройден
2	6	Пользователь меняет размер браузера или заходит с другого устройства	Пользовательский интерфейс сохраняет правильный вид (не разъезжаются кнопки, панель навигации читаема и функционирует как и на большом экране)	Пройден

Таблица 4.2 – Результаты тестирования пользовательского интерфейса

Номер тестового примера	Номер соответствующего тестового требования	Сценарий выполнения тестового примера (действие оператора)	Реакция системы	Результат (пройден / не пройден)
3	1	Перейти на страницу авторизации, нажав на кнопку LogIn в панели навигации будучи не авторизованным	Отображается форма авторизации	пройден
4	5	Пользователь вошёл в комнату будучи учеником	Кнопки создания материала не отображаются	пройден
5	5	Пользователь вошёл в комнату будучи преподавателем	Отображаются элементы создания новых материалов	пройден
6	4	Пользователь перешёл по ссылке в браузере в комнату, к которой у него нет доступа	Отобразилась страница с ошибкой 404	пройден
7	3	Пользователь перешёл в комнату	Вкладки с материалами и домашними заданиями разбиты на две колонки	пройден
8	2	Пользователь перешёл в список комнат	Пользователю отображаются комнаты и основная информация о них	пройден

4.1.3 Модульное тестирование

Для тестирования данного приложения был выбран фреймворк pytest для python, который позволяет удобно проводить модульное тестирование. Исходный код самих тестов представлен в Приложение Б. Вся главная логика сервисов приложения тестируется модульно, и разбита по разным файлам. Для распределения данных между тестами используются фикстуры. Специальный набор данных, предназначенный для тестирования, с имитацией исходных данных, приближенных к настоящим стохастическим переменным (например,

имитация данных пользователя). Для игнорирования долгих операций (например, отправка письма на электронную почту в тестах не нужна, ибо создаётся не существующий пользователь) используются mock-объекты, имитирующие вызовы функций и методов без непосредственного вызова. Результаты модульного тестирования представлены на рисунке 37

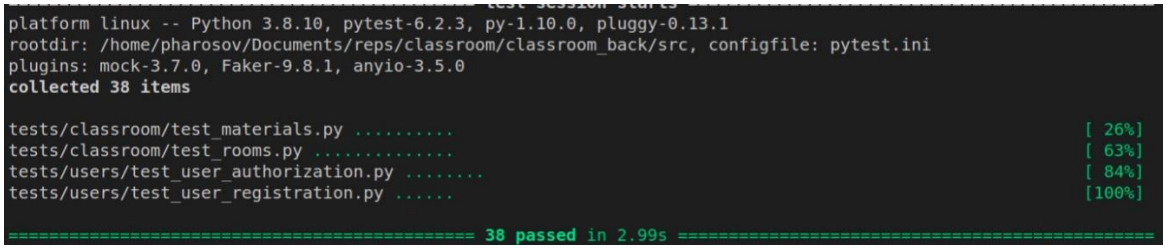


Рисунок 37 - Результаты модульного тестирования

Данные тесты обеспечивают покрытие 91% кода. Результаты анализа покрытия тестами приложения представлены на рисунке 38.

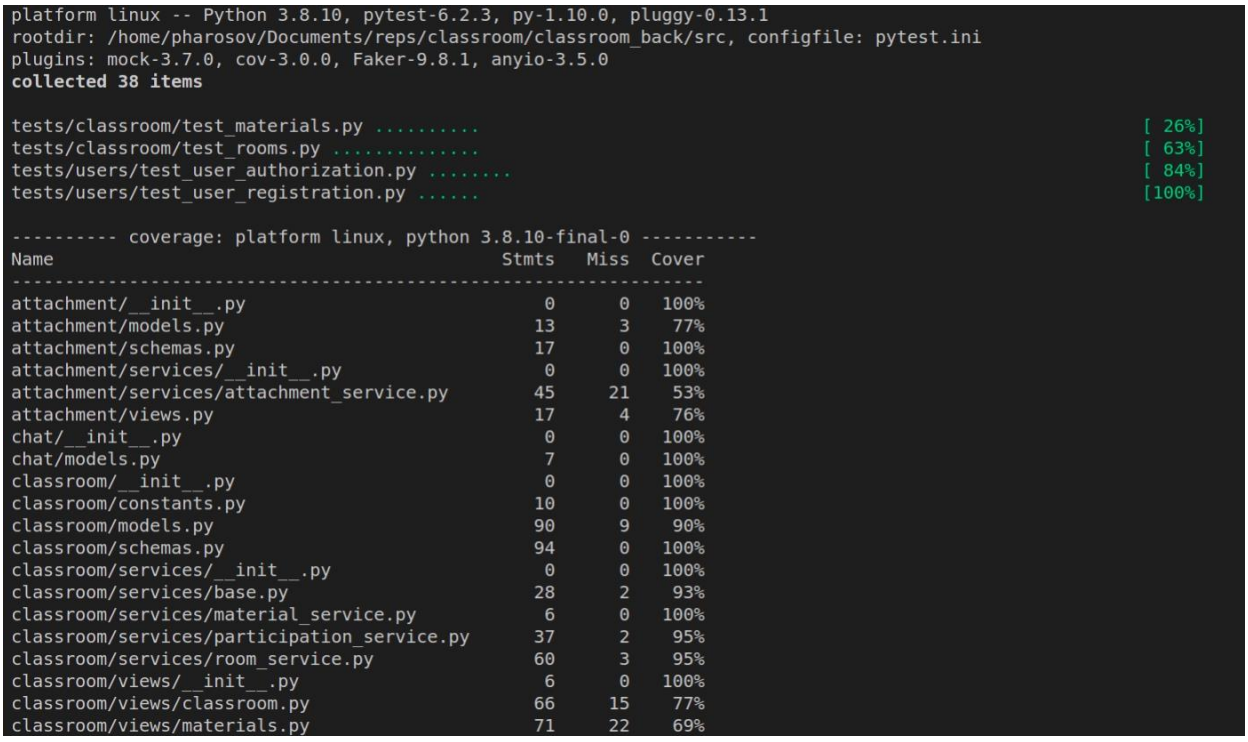
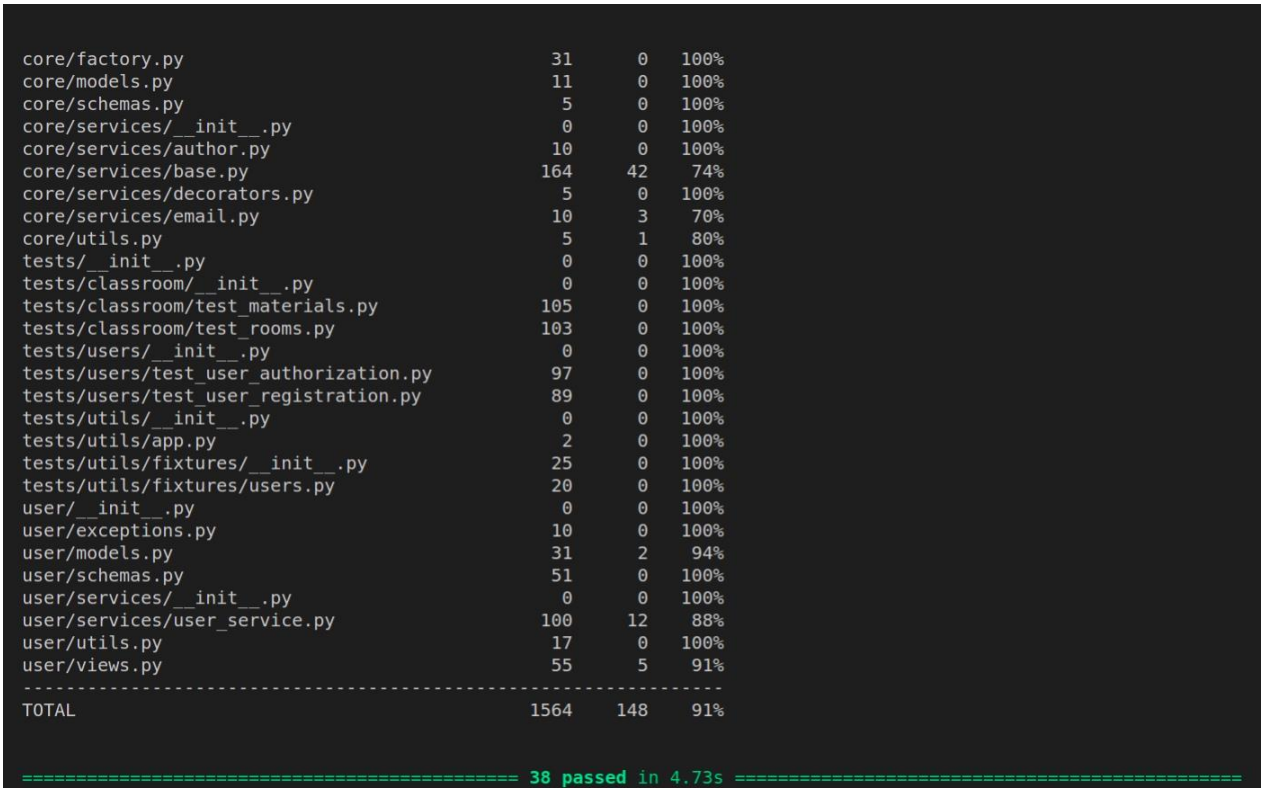


Рисунок 38 - Результаты анализа покрытия тестами приложения



```
core/factory.py                                31      0   100%
core/models.py                                 11      0   100%
core/schemas.py                                 5      0   100%
core/services/__init__.py                       0      0   100%
core/services/author.py                        10      0   100%
core/services/base.py                         164     42    74%
core/services/decorators.py                     5      0   100%
core/services/email.py                         10      3    70%
core/utils.py                                   5      1    80%
tests/__init__.py                               0      0   100%
tests/classroom/__init__.py                     0      0   100%
tests/classroom/test_materials.py             105      0   100%
tests/classroom/test_rooms.py                 103      0   100%
tests/users/__init__.py                         0      0   100%
tests/users/test_user_authorization.py         97      0   100%
tests/users/test_user_registration.py          89      0   100%
tests/utils/__init__.py                         0      0   100%
tests/utils/app.py                              2      0   100%
tests/utils/fixtures/__init__.py               25      0   100%
tests/utils/fixtures/users.py                  20      0   100%
user/__init__.py                                0      0   100%
user/exceptions.py                             10      0   100%
user/models.py                                 31      2    94%
user/schemas.py                                51      0   100%
user/services/__init__.py                       0      0   100%
user/services/user_service.py                 100     12    88%
user/utils.py                                  17      0   100%
user/views.py                                  55      5    91%
-----------------------------------------------------------------
TOTAL                                        1564    148    91%


============================================ 38 passed in 4.73s ============================================
```

Рисунок 38 (продолжение) - Результаты анализа покрытия тестами приложения

ЗАКЛЮЧЕНИЕ

Разработанный программный продукт является информационной системой обмена файлами и сообщениями для организации образовательного процесса. Разработанная система позволяет облегчать процесс обучения в удалённом формате для студентов и школьников, а также для людей, ведущих свои учебные курсы вне каких-либо учебных заведений.

В процессе создания системы в соответствии с заданием были разработаны: модель вариантов использования, концептуальная модель предметной области, диаграммы деятельности, реляционная модель данных, диаграмма состояний интерфейса, клиентская часть, диаграмма компонентов и диаграмма развертывания. Была выполнена частично проверка и отладка системы.

Данная система позволит:

– Организовать образовательный процесс

– Отслеживать статистику о процессе обучения

– Собрать все необходимые материалы для получения знаний по определённой дисциплине в одном месте для удобного пользования

– Идентифицировать каждого участника образовательного процесса, помимо ФИО выводить для преподавателя основную информацию по его успеваемости

– Сделать удобным и комфортным процесс обучения в дистанционном формате путём возможности постоянного поддерживания связи преподавателя с учениками

В данном курсовом проекте было проведено программирование на языке Python с использованием фреймворков Tortoise и FastAPI. Программный продукт содержит клиентскую часть для работы конечным клиентом в браузере и RestfulAPI, которое позволит в дальнейшем интегрировать его в любое другое приложение (например мобильное приложение) и позволит переиспользовать ту же логику, что реализована и на самом сайте.

Внедрение данной системы в образовательный процесс позволит сократить время, затрачиваемое на организацию самого процесса, автоматизировав и покрыв большую часть потребностей преподавателей. Кроме того, самим учащимся будет удобнее проходить обучение в удалённом формате, имея возможность постоянно поддерживать контакт с преподавателям в рамках конкретной учебной дисциплины и иметь постоянный доступ к учебным материалам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Python FastAPI Framework оффициальная документация [Электронный ресурс]: https://fastapi.tiangolo.com/ (дата обращения 10.02.22).

2. Сервис для построения электронных диаграмм [Электронный ресурс]: https://app.diagrams.net/ (дата обращения 4.10.2019).

3. PostgreSQL оффицаильное руководство по субд postgres [Электронный ресурс]: https://www.postgresql.org/docs/current/app-psql.html (дата обращения 4.11.2019).

4. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учеб. – М.: Финансы и статистика, 2000.

5. Python AsyncIO оффициальная документация [Электронный ресурс]: https://docs.python.org/3/library/asyncio.html (дата обращения 5.10.2019).

6. Tortoise ORM. Object relational mapping документация [Электронный ресурс]: https://tortoise-orm.readthedocs.io/en/latest/index.html (дата обращения 5.10.2019).

7. Bootstrap Guide. Руководство по фреймворку Bootstrap для клиентской части [Электронный ресурс]: https://tortoise-orm.readthedocs.io/en/latest/index.html (дата обращения 5.10.2019).

8. Мандел Т. Разработка пользовательского интерфейса. - М: ДМК Пресс, 2001.

9. Мартин Дж. Организация баз данных в вычислительных системах. - М.: Мир, 1980.

10. Чен П. Модель «сущность-связь» - шаг к единому представлению данных // СУБД. 1995. №3. С. 137-158. 11. Омельченко Л. Самоучитель Visual FoxPro 8.

11. Vue.js. Руководство по JavaScript фреймворку vue.js [Электронный ресурс]: https://v2.vuejs.org/v2/guide/?redirect=true (дата обращения 5.10.2019).

Приложение А

Текст программы

Запускающая программа app.py

```
from core.factory import AppFactory


app = AppFactory.create_app()
```

src/core/base.py

```
from typing import Dict, List, NewType, Tuple, Type, Union
from warnings import filters
from attr import attrs
from pydantic import BaseModel

from tortoise import models
from tortoise.transactions import atomic

from core.services.decorators import action


CreateSchema = NewType('CreateSchemaType', BaseModel)
UpdateSchema = NewType('UpdateSchemaType', BaseModel)
DeleteSchema = NewType('DeleteSchemaType', BaseModel)
ResultTuple = NewType('ResultTuple', Tuple[bool, Union[Dict, None]])


class SchemaMapMixin:
    schema_map: Dict = {}
    schema_class: Type[BaseModel] = None

    async def get_schema_class(self) -> Type[BaseModel]:
        return self.schema_map.get(self.action, self.schema_class)

    async def get_schema(self, **kwargs) -> BaseModel:
        '''
            Returns pydantic schema of the action if it is in `schema_map`
keys.
            Else returns async default `schema_class` instance.
        '''
        return await self.get_schema_class()(**kwargs)


class IServiceBase(SchemaMapMixin):
    model: Type[models.Model] = None
    _action_attributes: Dict = {}

    @property
    def current_action_attributes(self):
        return self._action_attributes.get(self.action, {})

    async def get_queryset(self, for_delete: bool = False):
        return self.model.all()

    error_messages = {
        'does_not_exist': 'Not found.',
        'no_values_found': 'No values were found {values}.',
        'already_exists': 'Already exists.'
    }
    required_fields_map = {}
    _errors: Dict = {}

    @action
    async def create(self, createSchema: CreateSchema, exclude_unset: bool
= False) -> Tuple[models.Model, Dict]:
        raise NotImplementedError()

    @action
```

```
    async def create(self, createSchema: CreateSchema, exclude_unset: bool
= False) -> Tuple[models.Model, Dict]:
        raise NotImplementedError()

    @action
    @atomic()
    async def bulk_update(
        self,
        updateSchema: UpdateSchema,
        filters: Dict = {},
        exclude_unset: bool = True,
    ) -> Tuple[models.Model, Dict]:
        raise NotImplementedError()


    @action
    @atomic()
    async def bulk_delete(
        self,
        deleteSchema: DeleteSchema,
        filters: Dict = {},
        exclude_unset: bool = True,
    ) -> Tuple[models.Model, Dict]:
        raise NotImplementedError()

    @action
    @atomic()
    async def bulk_create(
        self,
        listCreateSchema: List[CreateSchema],
        exclude_unset: bool = True,
    ):
        raise NotImplementedError()

    @action
    async def fetch(
        self,
        filters: Dict = {},
        ordering: List = [],
        distinct: bool = True,
    ):
        raise NotImplemented()

    @action
    async def retrieve(self, **kwargs):
        raise NotImplemented()


class CreateUpdateServiceMixin(IServiceBase):
    required_fields_map: Dict[str, List] = {}

    async def validate(self, attrs: Dict = {}) -> Dict:
        ''' Method for prevalidating attrs before action '''
        return attrs

    async def _validate_values(self, **kwargs) -> None:
        '''
            Validates values before creating or updating `model` field
instances.
        '''
```

```
        errors = {}
        self._action_attributes[self.action] = kwargs

        for  required_field  in  self.required_fields_map.get(self.action,
[]):
            if required_field not in kwargs:
                errors[required_field] = 'This field is required.'

        for key, value in kwargs.items():
            validation_method = f'validate_{key}'

            if hasattr(self, validation_method):
                validation_success, validation_error = await getattr(self,
validation_method)(value)

                if not validation_success:
                    errors[key] = validation_error

        attrs = await self.validate(kwargs) or kwargs
        return attrs, errors

    @action
    async def create(
        self,
        createSchema: CreateSchema,
        exclude_unset: bool = False,
        fetch_related: List[str] = [],
    ) -> Tuple[models.Model, Dict]:
        schema_dict = createSchema.dict(exclude_unset=exclude_unset)
        attrs, errors = await self._validate_values(**schema_dict)

        if errors:
            return None, errors
        obj = await self.model.create(**attrs)
        await obj.fetch_related(*fetch_related)
        return obj, {}

    @action
    @atomic
    async def bulk_create(
        self,
        listCreateSchema: List[CreateSchema],
    ):
        obj_list = []

        for createSchema in listCreateSchema:
            obj_list.append(
                await self.create(createSchema)
            )
        return obj_list

    @action
    async def update(
        self,
        id: int,
        updateSchema: UpdateSchema,
        exclude_unset: bool = True,
        fetch_related: List = [],
    ) -> Tuple[models.Model, Dict]:
        schema_dict = updateSchema.dict(exclude_unset=exclude_unset)
```

```
        attrs, errors = await self._validate_values(**schema_dict)

        if errors:
            return None, errors

        queryset = self.model.filter(id=id)
        await queryset.update(**attrs)
        obj = await queryset.first()

        if not obj:
            return None, self.error_messages.get('does_not_exist')

        await obj.fetch_related(*fetch_related)
        return obj, {}

    @action
    async def bulk_update(
        self,
        updateSchema: UpdateSchema,
        filters: Dict = {},
        exclude_unset: bool = True,
    ) -> Tuple[models.Model, Dict]:
        ''' Update multiple objects fitting passed filters '''
        schema_dict = updateSchema.dict(exclude_unset=exclude_unset)
        errors = await self._validate_values(**schema_dict)

        queryset = await self.model.filter(**filters)

        if not await queryset.exists():
            return None, { 'error': self.error_messages.get('does_not_exist') }

        await queryset.update(schema_dict)

        if errors:
            return None, errors
        return queryset, {}


class RetrieveFetchServiceMixin(IServiceBase):
    @action
    async def fetch(
        self,
        _filters: Dict = {},
        _ordering: List = [],
        _prefetch_related: List = [],
        _select_related: List = [],
        distinct: bool = True,
    ):
        qs = await self.get_queryset()
        queryset = qs.filter(**_filters).order_by(*_ordering)\
            .select_related(*_select_related)\
            .prefetch_related(*_prefetch_related)

        if distinct:
            queryset = queryset.distinct()
        return await queryset, None

    @action
    async def retrieve(
```

```
        self,
        _prefetch_related: list = [],
        _select_related: list = [],
        **filters,
    ):
        qs = await self.get_queryset()
        obj = qs.filter(**filters).prefetch_related(
            *_prefetch_related,
        ).select_related(*_select_related).first()

        if not obj:
            return None, self.error_messages.get('does_not_exist')
        return obj, None

    @action
    async def retrieve(
        self,
        _fetch_related: list = [],
        **kwargs,
    ):
        # TODO: чекнуть сколько запросов идёт в базу, скорее всего тут
лишние запросы идут
        qs = await self.get_queryset()
        obj                                =                                await
qs.filter(**kwargs).prefetch_related(*_fetch_related).first()

        if not obj:
            return None, self.error_messages.get('does_not_exist')
        return obj, {}


class DeleteMixin(IServiceBase):
    async def _validate_delete(self, attrs: Dict):
        return attrs, {}

    async def _validate_bulk_delete(self, **kwargs):
        qs = await self.get_queryset(for_delete=True)
        return await qs.filter(**kwargs).exists()

    @action
    async def delete(self, deleteSchema: DeleteSchema, exclude_unset: bool
= True):
        delete_schema_dict                                                  =
deleteSchema.dict(exclude_unset=exclude_unset)
        self._action_attributes[self.action] = delete_schema_dict
        attrs,               errors               =                 await
self._validate_delete(self.current_action_attributes)

        if errors:
            return None, errors
        return await self.model.filter(**attrs).delete(), None

    @action
    async def bulk_delete(
        self,
        **kwargs,
    ) -> Tuple[models.Model, Dict]:
        if not await self._validate_bulk_delete(**kwargs):
            return { 'permission_error': 'You are not allowed to do that!'
}
```

```
        qs = await self.get_queryset(for_delete=True)
        await qs.filter(**kwargs).delete()
        return None


class   CRUDService(CreateUpdateServiceMixin,   RetrieveFetchServiceMixin,
DeleteMixin):
    '''
        CrudService for performing CRUD operations and validation.
        ! All manager's operations only avaliable from that class and it's
subclasses methods!
    '''
    def __init__(self) -> None:
        self.user = None
```

src/classroom/views.py

```
from typing import List

from fastapi import APIRouter, Depends, Query
from fastapi.exceptions import HTTPException

from starlette import status

from classroom.schemas import (ParticipationCreateByJoinSlugSchema,
    ParticipationSuccessSchema,
    RoomCreateJoinLinkSuccessSchema,
    RoomCreateSchema,
    RoomCreateSuccessSchema, RoomDeleteSchema,
    RoomDetailSchema,
    RoomListItemSchema, RoomParticipationSchema,
)
from    classroom.services.room_service    import    ParticipationService,
RoomService

from user.models import User
from user.utils import get_current_user


classroom_router = APIRouter()

@classroom_router.post(
    '',
    response_model=RoomCreateSuccessSchema,
    status_code=status.HTTP_201_CREATED,
    operation_id='createRoom',
)
async def create_new_room(
    roomCreateSchema: RoomCreateSchema,
    user: User = Depends(get_current_user),
):

    room_service = RoomService(user)
    room, errors = await room_service.create(roomCreateSchema)

    if errors:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
```

```
            detail=errors,
        )

    return RoomCreateSuccessSchema(
        id=room.id,
        name=room.name,
        description=room.description,
    )


@classroom_router.put(
    '/{room_id}',
    response_model=RoomCreateSuccessSchema,
    status_code=status.HTTP_200_OK,
    operation_id='updateRoom',
)
async def update_room(
    roomUpdateSchema: RoomCreateSchema,
    room_id: int,
    user: User = Depends(get_current_user),
):
    room_service = RoomService(user)
    room, errors = await room_service.update(room_id, roomUpdateSchema)

    if errors:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=errors,
        )

    return RoomCreateSuccessSchema(
        id=room.id,
        name=room.name,
        description=room.description,
    )


@classroom_router.delete(
    '/{room_id}',
    status_code=status.HTTP_204_NO_CONTENT,
    operation_id='deleteRoom',
)
async def delete_room(
    room_id: int,
    user: User = Depends(get_current_user),
):
    roomDeleteSchema = RoomDeleteSchema(id=room_id)
    room_service = RoomService(user)
    deletes_count, errors = await room_service.delete(roomDeleteSchema)

    if errors:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=errors,
        )
    if not deletes_count:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
        )
```

```
@classroom_router.get(
    '/{room_id}',
    response_model=RoomDetailSchema,
    operation_id='getRoom',
)
async def get_room(
    room_id: int,
    user: User = Depends(get_current_user),
):
    room_service = RoomService(user)
    room, errors = await room_service.retrieve(
        _fetch_related=['participations', 'author'],
        id=room_id,
    )

    if errors:
        raise         HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail=errors)
    return RoomDetailSchema.from_orm(room)


@classroom_router.get(
    '',
    response_model=List[RoomListItemSchema],
    operation_id='getCurrentUserRooms',
)
async def current_user_room_list(
    user: User = Depends(get_current_user),
    ordering: List[str] = Query(['-created_at']),
):
    room_service = RoomService(user)
    rooms, errors = await room_service.fetch(_ordering=ordering)

    if errors:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=errors,
        )

    room_response_list = []

    # говнокод, потом переделать
    for room in rooms:
        await room.fetch_related('participations')
        participation, _ = await room_service.participation_service.fetch({
            'room_id': room.id,
            'user_id': user.id,
        })
        participation = participation[0]

        room_response_list.append(
            RoomListItemSchema(
                id=room.id,
                name=room.name,
                description=room.description,
                participations_count=room.participations_count,
                participation=RoomParticipationSchema(
                    id=participation.id,
                    role=participation.role.name,
```

```
                    user_id=participation.user_id,
                    room_id=participation.room_id,
                    created_at=participation.created_at,
                ),
                created_at=room.created_at,
            )
        )
    return room_response_list


@classroom_router.post(
    '/{room_id}/refresh_join_slug',
    response_model=RoomCreateJoinLinkSuccessSchema,
)
async def refresh_join_slug(
    room_id: int,
    user: User = Depends(get_current_user),
):
    room_service = RoomService(user)
    join_slug, errors = await room_service.refresh_join_slug(room_id)

    if errors:
        raise       HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=errors)
    return RoomCreateJoinLinkSuccessSchema(join_slug=join_slug)


@classroom_router.get(
    '/join/{join_slug}',
    response_model=ParticipationSuccessSchema,
)
async def join_room_by_link(
    join_slug: str,
    user: User = Depends(get_current_user),
):
    participation_service = ParticipationService(user)

    participation, errors = await participation_service.create(
        ParticipationCreateByJoinSlugSchema(
            user_id=user.id,
            join_slug=join_slug,
            author_id=user.id,
        ),
        fetch_related=['room', 'room__author']
    )

    if errors:
        raise       HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail=errors)

    return ParticipationSuccessSchema(
        id=participation.id,
        user_id=participation.user_id,
        role=participation.role.name,
        author_id=participation.author_id,
        room=RoomListItemSchema.from_orm(participation.room),
    )
```

Приложение Б

Исходный код модульных тестов приложения

test_materials.py

```
import asyncio
import hashlib
from faker import Faker
from fastapi.applications import FastAPI
from fastapi_jwt_auth import AuthJWT

from fastapi import status
from fastapi.testclient import TestClient
import pytest
from classroom.constants import ParticipationRoleEnum

from classroom.models import Material, Room, Participation

from tests.utils.fixtures import client, event_loop, fake, app
from tests.utils.fixtures.users import authentication_token

from user.models import User


@pytest.fixture
def room(
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
):
    user= event_loop.run_until_complete(User.first())
    room, _ = event_loop.run_until_complete(Room.get_or_create(
        defaults={
            'id': 1,
            'name': fake.text(),
            'description': fake.text()[:50],
            'text': fake.text(),
            'join_slug': fake.md5(),
            'author': user,
            'updated_by': user,
        }
    ))
    event_loop.run_until_complete(Participation.get_or_create(
        defaults={
            'id': 1,
            'role': ParticipationRoleEnum.host,
            'room': room,
            'author': user,
            'user': user,
            'updated_by': user,
        }
    ))
    event_loop.run_until_complete(room.fetch_related('materials'))
```

```
    yield room


@pytest.fixture
def material(
    event_loop: asyncio.AbstractEventLoop,
):
    material = event_loop.run_until_complete(Material.first())
    yield material


def test_material_create_success(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    material: Material,
):
    url = app.url_path_for('create_new_material')

    assert room
    assert not material

    response = client.post(url, json={
        'title': 'fake name',
        'description': 'test description',
        'room_id': room.id,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_201_CREATED,
response.json()
    assert event_loop.run_until_complete(Material.first())


def test_material_create_not_logged_in(
    app: FastAPI,
    client: TestClient,
    room: Room,
):
    url = app.url_path_for('create_new_material')

    assert room

    response = client.post(url, json={
        'title': 'fake name',
        'description': 'test description',
        'room_id': room.id,
    })

    assert response.status_code == status.HTTP_401_UNAUTHORIZED,
response.json()
```

```
def test_material_create_not_a_moder(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    authentication_token: str,
):
    url = app.url_path_for('create_new_material')

event_loop.run_until_complete(Participation.filter(room_id=room.id)
\
        .update(role=ParticipationRoleEnum.participant))

    assert room

    response = client.post(url, json={
        'title': 'fake name',
        'description': 'test description',
        'room_id': room.id,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_400_BAD_REQUEST,
response.json()


def test_material_get(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):
    url = app.url_path_for('get_materials')
    assert len(room.materials)

    response = client.get(
        url,
        params={
            'room_id': room.id,
        }, headers={
            'Authorization': f'Bearer {authentication_token}'
        }
    )

    event_loop.run_until_complete(room.refresh_from_db())
    assert response.status_code == status.HTTP_200_OK,
response.json()
    materials = response.json()['items']
    assert len(materials) == len(room.materials)


def test_material_get_not_a_moder(
```

```
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    material: Material,
    authentication_token: str,
):
    url = app.url_path_for('get_materials')

event_loop.run_until_complete(Participation.filter(room_id=room.id)
\
        .update(role=ParticipationRoleEnum.participant))

    assert room

    response = client.get(
        url,
        params={
            'room_id': room.id,
        }, headers={
            'Authorization': f'Bearer {authentication_token}'
        }
    )

    event_loop.run_until_complete(room.refresh_from_db())
    assert     response.status_code     ==     status.HTTP_200_OK,
response.json()
    materials = response.json()['items']
    assert len(materials) == len(room.materials)


def test_material_get_not_in_room(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    authentication_token: str,
):
    url = app.url_path_for('get_materials')
    event_loop.run_until_complete(Participation.all().delete())

    assert room

    response = client.get(
        url,
        params={
            'room_id': room.id,
        }, headers={
            'Authorization': f'Bearer {authentication_token}'
        }
    )

    event_loop.run_until_complete(room.refresh_from_db())
    assert   response.status_code   ==   status.HTTP_403_FORBIDDEN,
response.json()
```

```
def test_update_material_success(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    material: Material,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.host))
    url             =             app.url_path_for('update_material',
material_id=material.id)

    new_material_title = 'Updated title'
    new_material_description = 'Updated description'
    new_material_text = 'updated text'

    response = client.put(url, json={
        'title': new_material_title,
        'description': new_material_description,
        'text': new_material_text,
        'room_id': room.id,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert      response.status_code      ==      status.HTTP_200_OK,
response.json()
    event_loop.run_until_complete(material.refresh_from_db())

    assert material.title == new_material_title
    assert material.description == new_material_description
    assert material.text == new_material_text


def test_update_material_moderator(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    material: Material,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.moderator))
    url             =             app.url_path_for('update_material',
material_id=material.id)

    new_material_title = 'Updated title2'
    new_material_description = 'Updated description2'
    new_material_text = 'updated text2'
```

```
    response = client.put(url, json={
        'title': new_material_title,
        'description': new_material_description,
        'text': new_material_text,
        'room_id': room.id,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_200_OK,
response.json()
    event_loop.run_until_complete(material.refresh_from_db())

    assert material.title == new_material_title
    assert material.description == new_material_description
    assert material.text == new_material_text


def test_update_material_participant(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
    material: Material,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.participant))
    url = app.url_path_for('update_material',
material_id=material.id)

    new_material_title = 'Updated title2'
    new_material_description = 'Updated description2'
    new_material_text = 'updated text2'

    response = client.put(url, json={
        'title': new_material_title,
        'description': new_material_description,
        'text': new_material_text,
        'room_id': room.id,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_400_BAD_REQUEST,
response.json()


def test_update_room_not_logged_in(
    app: FastAPI,
    client: TestClient,
    material: Material,
    room: Room,
```

```
):
    url             =             app.url_path_for('update_material',
material_id=material.id)

    new_material_title = 'Updated title2'
    new_material_description = 'Updated description2'
    new_material_text = 'updated text2'

    response = client.put(url, json={
        'title': new_material_title,
        'description': new_material_description,
        'text': new_material_text,
        'room_id': room.id,
    })
    assert  response.status_code  ==  status.HTTP_401_UNAUTHORIZED,
response.json()

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.host))
```

test_rooms.py

```
import asyncio
import hashlib
from faker import Faker
from fastapi.applications import FastAPI
from fastapi_jwt_auth import AuthJWT

from fastapi import status
from fastapi.testclient import TestClient
import pytest
from classroom.constants import ParticipationRoleEnum

from classroom.models import Room, Participation

from tests.utils.fixtures import client, event_loop, fake, app
from tests.utils.fixtures.users import authentication_token

from user.models import User


@pytest.fixture
def room(
    event_loop: asyncio.AbstractEventLoop
):
    yield event_loop.run_until_complete(Room.first())


def test_room_create_success(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
```

```
):
    url = app.url_path_for('create_new_room')

    assert not room

    response = client.post(url, json={
        'name': 'fake name',
        'description': 'test description',
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_201_CREATED
    assert event_loop.run_until_complete(Room.first())


def test_room_create_not_logged_in(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
):
    url = app.url_path_for('create_new_room')

    response = client.post(url, json={
        'name': 'fake name',
        'description': 'test description',
    }, headers={
        'Authorization': f'Bearer fake_token'
    })

    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_room_refresh_invite_link(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):
    url = app.url_path_for('refresh_join_slug', room_id=room.id)
    previous_join_slug = room.join_slug

    response = client.post(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    event_loop.run_until_complete(room.refresh_from_db())
    assert response.status_code == status.HTTP_200_OK
    assert previous_join_slug != room.join_slug


def test_join_room_by_link_success(
    authentication_token: str,
```

```
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):
    event_loop.run_until_complete(Participation.all().delete())
    url           =           app.url_path_for('join_room_by_link',
join_slug=room.join_slug)

    response = client.get(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    participation                                                  =
event_loop.run_until_complete(Participation.filter(room_id=room.id)
.first())
    assert response.status_code == status.HTTP_200_OK
    assert participation.room_id == room.id
    assert participation.role == ParticipationRoleEnum.participant


def test_join_room_by_link_already_in_room(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):
    url           =           app.url_path_for('join_room_by_link',
join_slug=room.join_slug)
    participations_count                                           =
event_loop.run_until_complete(Participation.all().count())

    response = client.get(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert                  participations_count                  ==
event_loop.run_until_complete(Participation.all().count())


def test_join_room_by_link_not_logged_in(
    app: FastAPI,
    client: TestClient,
    room: Room,
):
    url           =           app.url_path_for('join_room_by_link',
join_slug=room.join_slug)

    response = client.get(url)
    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_update_room_success(
```

```
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.host))
    url = app.url_path_for('update_room', room_id=room.id)

    new_room_name = 'Updated name'
    new_room_description = 'Updated description'
    response = client.put(url, json={
        'name': new_room_name,
        'description': new_room_description,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_200_OK
    event_loop.run_until_complete(room.refresh_from_db())

    assert room.name == new_room_name
    assert room.description == new_room_description


def test_update_room_moderator(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.moderator))
    url = app.url_path_for('update_room', room_id=room.id)

    new_room_name = 'Updated name'
    new_room_description = 'Updated description'
    response = client.put(url, json={
        'name': new_room_name,
        'description': new_room_description,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_200_OK
    event_loop.run_until_complete(room.refresh_from_db())

    assert room.name == new_room_name
    assert room.description == new_room_description
```

```
def test_update_room_participant(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.participant))
    url = app.url_path_for('update_room', room_id=room.id)

    new_room_name = 'Updated name'
    new_room_description = 'Updated description'
    response = client.put(url, json={
        'name': new_room_name,
        'description': new_room_description,
    }, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_400_BAD_REQUEST


def test_update_room_not_logged_in(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.participant))
    url = app.url_path_for('update_room', room_id=room.id)

    new_room_name = 'Updated name'
    new_room_description = 'Updated description'
    response = client.put(url, json={
        'name': new_room_name,
        'description': new_room_description,
    })

    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_delete_room_participant(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):
```

```
event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.participant))
    url = app.url_path_for('delete_room', room_id=room.id)

    response = client.delete(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_400_BAD_REQUEST


def test_delete_room_not_logged_in(
    app: FastAPI,
    client: TestClient,
    room: Room,
):
    url = app.url_path_for('delete_room', room_id=room.id)

    response = client.delete(url)
    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_room_get_detail(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    room: Room,
):
    url = app.url_path_for('get_room', room_id=room.id)

    response = client.get(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_200_OK


def test_delete_room_success(
    authentication_token: str,
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    room: Room,
):

event_loop.run_until_complete(Participation.filter(room_id=room.id)
.update(role=ParticipationRoleEnum.host))
    url = app.url_path_for('delete_room', room_id=room.id)

    response = client.delete(url, headers={
        'Authorization': f'Bearer {authentication_token}'
    })

    assert response.status_code == status.HTTP_204_NO_CONTENT
```

test_user_authorization.py

```
import asyncio
from datetime import datetime

import hashlib
from typing import Dict
from faker import Faker
from fastapi.applications import FastAPI

from fastapi import status
from fastapi.testclient import TestClient
import pytest

from tests.utils.fixtures import client, event_loop, fake, app

from user.models import User


USER_TEST_PASSWORD = 'testpPassword123'
USER_TEST_UPDATE_PASSWORD = 'testpAssword321'


@pytest.fixture(scope='module')
def user_data(
    fake: Faker
):
    yield {
        'id': 1,
        'first_name': fake.name(),
        'last_name': fake.name(),
        'email': fake.email(),
        'password':
hashlib.md5(USER_TEST_PASSWORD.encode()).hexdigest(),
        'phone_number': '+79990001122',
        'activation_token': 'test_activation_token',
        'activation_deadline_dt': datetime.utcnow(),
    }


@pytest.fixture
def user(
    event_loop: asyncio.AbstractEventLoop,
    user_data: Dict,
) -> User:
    user = event_loop.run_until_complete(
        User.get_or_create(
            defaults={**user_data, 'is_active': False,},
        )
    )[0]
    yield user
```

```
def test_authentication_fail_not_active(
    user: User,
    app: FastAPI,
    client: TestClient,
    user_data: Dict,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'email': user.email,
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_401_UNAUTHORIZED
    assert response.json()['detail'] == 'User is not active. Please
activate your profile.'


def test_user_activation(
    user: User,
    app: FastAPI,
    client: TestClient,
    user_data: Dict,
    event_loop: asyncio.AbstractEventLoop,
):
    url              =              app.url_path_for('activate_user',
activation_token=user_data['activation_token'])
    response = client.get(url)


def test_authentication_success_email(
    user: User,
    app: FastAPI,
    client: TestClient,
    user_data: Dict,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'email': user.email,
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    assert 'access_token' in json_data
    assert 'refresh_token' in json_data
```

```
    access_token = json_data['access_token']

    url = app.url_path_for('current_user_info')

    response = client.get(url, headers={
        'Authorization': f'Bearer {access_token}'
    })

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    for key in list(json_data.keys()):
        if key in user_data:
            assert user_data[key] == json_data[key], key


def test_authentication_success_phone(
    user: User,
    app: FastAPI,
    client: TestClient,
    user_data: Dict,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'phone_number': user.phone_number,
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    assert 'access_token' in json_data
    assert 'refresh_token' in json_data

    access_token = json_data['access_token']

    url = app.url_path_for('current_user_info')

    response = client.get(url, headers={
        'Authorization': f'Bearer {access_token}'
    })

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    for key in list(json_data.keys()):
        if key in user_data:
            assert user_data[key] == json_data[key], key
```

```
def test_authentication_fail_phone(
    app: FastAPI,
    client: TestClient,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'phone_number': '+0000000000',
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_authentication_fail_email(
    app: FastAPI,
    client: TestClient,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'email': 'notauseremail@notamail.com',
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_401_UNAUTHORIZED


def test_any_user_info(
    user: User,
    app: FastAPI,
    client: TestClient,
    user_data: Dict,
):
    url = app.url_path_for('authenticate_user')

    response = client.post(
        url,
        json={
            'email': user.email,
            'password': USER_TEST_PASSWORD,
        }
    )

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    assert 'access_token' in json_data
    assert 'refresh_token' in json_data
```

```
    access_token = json_data['access_token']

    url = app.url_path_for('user_info', user_id=user.id)

    response = client.get(url, headers={
        'Authorization': f'Bearer {access_token}'
    })

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    for key in list(json_data.keys()):
        if key in user_data:
            assert user_data[key] == json_data[key], key


def test_current_user_update(
    user: User,
    app: FastAPI,
    client: TestClient,
    fake: Faker,
    user_data: Dict,
):
    url = app.url_path_for('authenticate_user')

    new_user_data = {
        'email': fake.email(),
        'first_name': fake.name(),
        'last_name': fake.name(),
        'password': USER_TEST_UPDATE_PASSWORD,
        'repeat_password': USER_TEST_UPDATE_PASSWORD,
        'confirm_password': USER_TEST_PASSWORD
    }

    response = client.post(
        url,
        json={
            'email': user.email,
            'password': USER_TEST_PASSWORD,
        },
    )

    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    assert 'access_token' in json_data
    assert 'refresh_token' in json_data

    access_token = json_data['access_token']

    url = app.url_path_for('update_current_user')

    response = client.put(url, json=new_user_data,
headers={'Authorization': f'Bearer {access_token}'})
```

```
    assert response.status_code == status.HTTP_200_OK
    json_data = response.json()

    for key in list(json_data.keys()):
        if key in new_user_data:
            assert new_user_data[key] == json_data[key]
```

test_user_registration.py

```
import asyncio
import hashlib
from faker import Faker
from fastapi.applications import FastAPI

from fastapi import status
from fastapi.testclient import TestClient
import pytest

from tests.utils.fixtures import client, event_loop, fake, app

from user.models import User


@pytest.fixture(scope='session')
def phone_number(fake: Faker) -> str:
    yield f'+{fake.random_int(1 * 10*11, int("9" * 13))}'

@pytest.fixture(scope='session')
def email(fake: Faker) -> str:
    yield fake.email()


def test_user_registration_success(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    phone_number: str,
    email: str,
):
    url = app.router.url_path_for('register_user')
    password = 'KjoiunslAdjkl19'

    user_count = event_loop.run_until_complete(User.all().count())

    assert not user_count

    user_creds = {
        'first_name': 'Валерий',
        'last_name': 'Жмышенко',
        'middle_name': 'Альбретович',
        'phone_number': phone_number,
        'password': password,
        'repeat_password': password,
```

```
        'accept_eula': True,
        'email': email,
    }

    response = client.post(url, json=user_creds)

    assert    response.status_code    ==    status.HTTP_201_CREATED,
response.json()
    assert    event_loop.run_until_complete(User.all().count())    ==
user_count + 1

    user = event_loop.run_until_complete(User.all().first())

    assert user.first_name == user_creds['first_name']
    assert user.last_name == user_creds['last_name']
    assert user.middle_name == user_creds['middle_name']
    assert                        user.password                        ==
hashlib.md5(password.encode()).hexdigest()
    assert user.activation_token
    assert user.created_at.date()


def test_user_registration_phone_and_email_taken(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
):
    url = app.router.url_path_for('register_user')
    password = 'Kjoisun41241kl19'

    user_count = event_loop.run_until_complete(User.all().count())
    user = event_loop.run_until_complete(User.all().first())

    assert user_count

    user_creds = {
        'first_name': fake.name(),
        'last_name': fake.name(),
        'middle_name': fake.name(),
        'phone_number': user.phone_number,
        'password': password,
        'repeat_password': password,
        'accept_eula': True,
        'email': user.email,
    }

    response = client.post(url, json=user_creds)
    json_data = response.json()

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert  json_data['detail']['phone_number']  ==  'User  with  that
phone number is already registred.'
    assert json_data['detail']['email'] == 'User with that email is
already registred.'
```

```
    assert   event_loop.run_until_complete(User.all().count())   ==
user_count


def test_user_registration_eula_is_not_accepted(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
):
    url = app.router.url_path_for('register_user')
    password = 'Kjoisun41241kl19'

    user_count = event_loop.run_until_complete(User.all().count())

    assert user_count

    user_creds = {
        'first_name': fake.name(),
        'last_name': fake.name(),
        'middle_name': fake.name(),
        'phone_number': '+79997078922',
        'email': 'jma@mail.ru',
        'password': password,
        'repeat_password': password,
        'accept_eula': False,
    }

    response = client.post(url, json=user_creds)
    json_data = response.json()

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert json_data['detail']['accept_eula'] == 'Please accept eula
and try again.'


def test_user_registration_eula_is_not_passed(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
):
    url = app.router.url_path_for('register_user')
    password = 'Kjoisun41241kl19'

    user_count = event_loop.run_until_complete(User.all().count())

    assert user_count

    user_creds = {
        'first_name': fake.name(),
        'last_name': fake.name(),
        'middle_name': fake.name(),
        'phone_number': '+79997078922',
        'email': 'jma@mail.ru',
```

```
        'password': password,
        'repeat_password': password,
    }

    response = client.post(url, json=user_creds)
    json_data = response.json()

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert json_data['detail']['accept_eula'] == 'Please accept eula
and try again.'


def test_user_registration_passwords_dont_match(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
    email: str,
):
    url = app.router.url_path_for('register_user')
    password = 'Kjoisun41241kl19'

    user_count = event_loop.run_until_complete(User.all().count())

    assert user_count

    user_creds = {
        'first_name': fake.name(),
        'last_name': fake.name(),
        'middle_name': fake.name(),
        'phone_number': '+79997078922',
        'password': password + '123',
        'email': 'jma@mail.ru',
        'repeat_password': password,
        'accept_eula': True,
    }

    response = client.post(url, json=user_creds)
    json_data = response.json()

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert  json_data['detail']['password']  ==  "Passwords  don't
match."


def test_user_registration_multiple_errors(
    app: FastAPI,
    client: TestClient,
    event_loop: asyncio.AbstractEventLoop,
    fake: Faker,
):
    url = app.router.url_path_for('register_user')
    password = 'Kjoisun41241kl19'

    user_count = event_loop.run_until_complete(User.all().count())
```

```
    user = event_loop.run_until_complete(User.all().first())

    assert user_count

    user_creds = {
        'first_name': fake.name(),
        'last_name': fake.name(),
        'middle_name': fake.name(),
        'phone_number': user.phone_number,
        'email': 'jma@mail.ru',
        'password': password + '123',
        'repeat_password': password,
        'accept_eula': False,
    }

    response = client.post(url, json=user_creds)
    json_data = response.json()

    assert response.status_code == status.HTTP_400_BAD_REQUEST
    assert  json_data['detail']['password']  ==  "Passwords  don't
match."
    assert  json_data['detail']['phone_number']  ==  "User  with  that
phone number is already registred."
    assert json_data['detail']['accept_eula'] == "Please accept eula
and try again."
```