1 Разработка средства обмена сообщениями

Чтобы в течение игрового процесса игроки могли взаимодействовать между собой, было принято решение разработать средство обмена сообщениями (далее «чат»). Чат позволяет игрокам совместно решать поставленные задачи, а также обмениваться опытом за короткий промежуток времени.

Чаты делятся на три типа:

— общий - это такой чат, который по умолчанию есть у всех пользователей, из него нельзя выйти, в нем состоят абсолютно все зарегистрированные пользователи, и владельцем этого чата является администратор;

— групповой - это такой чат, который состоит из трех и более человек и владельцем которого является либо создатель чата, либо тот человек, которому были выданы соответствующие права;

— приватный - это такой чат, который состоит из двух человек.

Для разработки чата потребовалось дополнить существующую базу данных, а также продумать процесс работы чата и реализовать соответствующие методы.

В процессе проектирования таблиц базы данных была создана схема, изображенная на рисунке 1.1

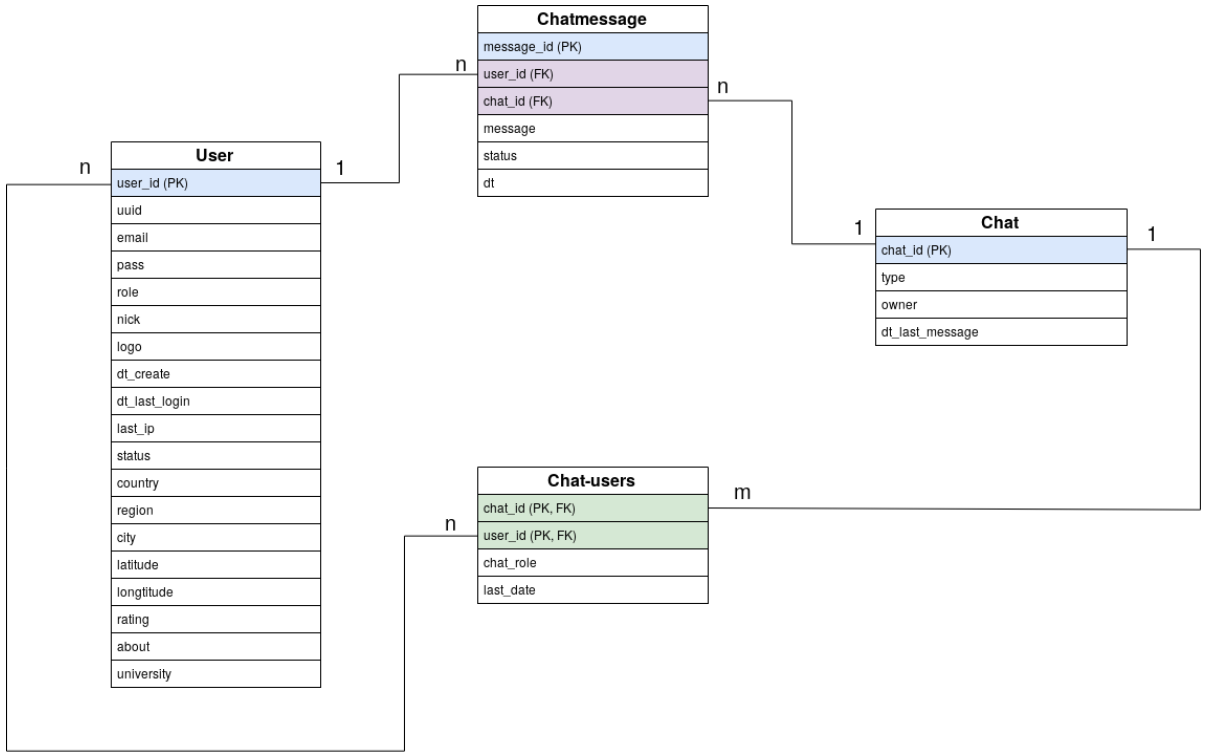

Рисунок 1.1 – Схема базы данных

Далее приведено описание таблиц и их полей.

Таблица «Chatmessages» нужна для того, чтобы хранить всю необходимую информацию о сообщениях, и имеет следующие поля:

– message_id - хранит идентификатор сообщения;

– user_id - идентификатор пользователя, автор сообщения;

– chat_id - идентификатор чата, в который было отправлено сообщение;

– message - текст сообщения;

– status - статус сообщение: "прочитано"или "не прочитано";

– dt - дата и время отправления сообщения.

Таблица «User» нужна для того, чтобы хранить информацию о пользователе, и имеет следующие поля:

– user_id - идентификатор пользователя;

– uuid - дополнительный строковый идентификатор;

– email - почтовый адрес пользователя;

– pass - пароль пользователя;

– nick - псевдоним, использующийся пользователем в системе;

– logo - графическое изображения пользователя;

– dt_create - дата создания пользователя;

– dt_last_login - время последней активности пользователя;

– last_ip - последний IP-адрес, с которого заходил пользователя;

– status - статус пользователя: в сети он или нет;

– country - страна проживания пользователя;

– region - регион проживания пользователя;

– city - город проживания пользователя;

– latitude - географическая широта;

– longtitute - географическая долгота;

– rating - рейтинг пользователя: количество набранных очков.

Таблица «Chat» нужна для того, чтобы хранить информацию о чате. Имеет следующие поля:

– chat_id - идентификатор чата;

– type - тип чата: общий, групповой или приватный;

– owner - владелец чата. Изначально им является создатель, но может быть переназначен;

– dt_last_message - дата последнего отправленного сообщения в чат.

Таблица «Chat-users» реализует связь «многие-ко-многим» между таблицами «Chat» и «User». Нужна для того, чтобы у пользователей была возможность состоять в нескольких чатах, и чтобы каждый чат мог содержать от двух до неопределенного количества пользователей. Содержит следующие поля:

– chat_id - идентификатор чата;

– user_id - идентификатор пользователя;

– chat_role - роль пользователя в чате.

После создания таблиц и их добавления в базу данных был предложен перечень методов для реализации чата, а именно:

– создание диалога: иниициализация диалога пользователем в начале об-

щения с другим пользователем или создание группового чата.

— отправить сообщения: отправление сообщения в текущий чат;

— прочитать сообщение: получение списка всех сообщений в текущем чате, а также меняет их статус с «не прочитано» на «прочитано»;

— показать список диалогов: получение списка всех чатов, в которых состоит пользователь;

— редактировать сообщение: исправление текста сообщения, которое уже было отправлено в чат;

— удаление сообщения: удаление сообщение из чата, в который оно было отправлено;

— ответить на сообщение: выделение сообщения или нескольких сообщений, которые можно переслать с комментарием;

— добавить в групповой чат: добавление пользователей в групповой чат администратором;

— исключить из группового чата: удаление пользователя из чата администратором чата или же самим пользователем, если он решил покинуть чат;

— смена владельца чата: передача прав администрирования чата другому пользователю;

— добавить в черный список: добавление в «черный список» пользователей, от которых другой пользователь не хочет получать сообщения;

— удалить из черного списка: удаление пользователя из черного списка с последующим возвращением прав писать сообщения тому, кто ранее их туда добавил.

Для разработки данных методов применялась методология разработки программного обеспечения TDD (Test-Driven Development), которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам. Данная методология полезна еще и тем, что на данном этапе веб-интерфейс сайта не обладает достаточным функционалом для тестирования

разрабатываемых методов.

Всего было реализовано четыре метода из вышеперечисленных, а именно: отправка, чтение, редактирование и удаление сообщения. По методологии TDD были написаны тесты, а затем сами методы.

Результат работы представлен на рисунке 1.2

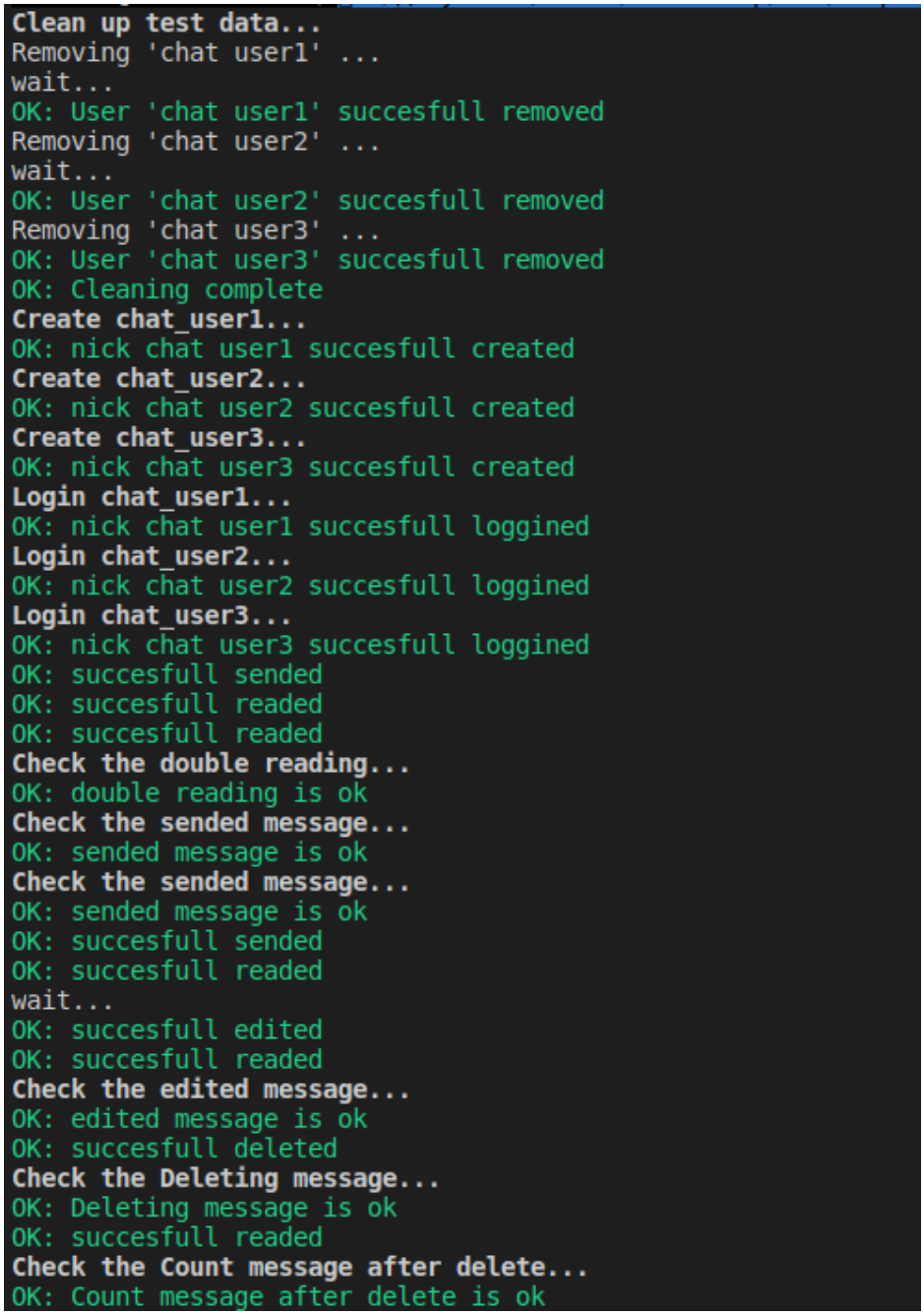

Рисунок 1.2 – Результат разработки методов для чата по методологии TDD

Эти тесты работают следующим образом: вызывается некоторый метод

из API (Application User Interface) с определенными параметрами. Возвращаемое значение метода известно заранее, поэтому, исходя из этого, можно проводить сравнение полученное значение с ожидаемым. Если они совпадают, тест пройден успешно, если нет, то, соответственно, тест не пройден.

Рассмотрим один из вызовов тестируемого метода на примере метода chats_send_message (отправка сообщения). На рисунке 1.3 реализован тест выходных данных на их соответствие.

```
def send_msg(session, req):
    res = session.chats_message_send(req)

    fhqtest.check_response(res, "succesfull sended")

    fhqtest.alert(res['message'] != req['message'], '"message" not equal')
    fhqtest.alert('user_id' not in res, 'Not member "user_id"')
    fhqtest.alert('dt' not in res, 'Not member "dt"')
    fhqtest.alert(res['dt'] == '', '"dt" empty')
    fhqtest.alert('message_id' not in res, 'Not member "message_id"')
    fhqtest.alert('status' not in res, 'Not member "status"')

    return res
```

Рисунок 1.3 – Тест выходных параметров

Ожидается, что функция chats_message_send вернет ответ, который состоит из: отправляемого сообщения, идентификатора пользователя-автора сообщения, времени отправки, идентификатора сообщения и его статуса.

Тест взаимосвязанных вызовов представлен на рисунке 1.4.

```
send_out_1 = send_msg(chat_user1_session, {
    "message": "hello",
    "chat": "0"
})

read_out_2 = read_msg(chat_user2_session, {
    "chat": "0"
})

read_out_3 = read_msg(chat_user3_session, {
    "chat": "0"
})

fhqtest.check_values("double reading", read_out_3['messages'], read_out_2['messages'])

fhqtest.check_values("sended message", read_out_3['messages'][-1]['message'], send_out_1['message'])
fhqtest.check_values("sended message", read_out_2['messages'][-1]['message'], send_out_1['message'])
```

Рисунок 1.4 – Тест выходных параметров

Ожидается, что последнее отправленное сообщение будет тем же самым, что и последнее полученное.

1.1 Angular

Для выполнения работы потребовалось с фреймворком Angular по причине того, что веб-интерфейс пользователя был написан с использованием этой технологии.

Angular - это TypeScript-фреймворк с открытым исходным кодом, который предназначен для разработки одностраничных приложений. Цель фреймворка - расширение браузерных приложений на основе MVC-шаблона, а также упрощение разработки и тестирования.

Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные TypeScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

Базовой единицей в данном фреймворке является компонент, который содержит HTML шаблон, стили и логику в TypeScript файлах.

1.2 Markdown

Markdown — облегчённый язык разметки, созданный с целью написания наиболее читаемого и удобного для правки текста, но пригодного для преобразования в языки для продвинутых публикаций.

Markdown используется как язык разметки для статей базы знаний.

2 Расширение фунционала «Базы знаний»

2.1 Изучение предметной области

База знаний в данной системе представлена списком статей в формате Markdown, который упрощает форматирование текста. Изначально добавлять, редактировать статьи и их перевод мог только администратор, что наклады-

вало на базу ряд ограничений: трудность поддержки актуальности статей, их качественное и количественное содержание.

Поэтому было решено разрешить пользователям предлагать свои правки и переводы и, соответственно, добавить этот функционал в веб-интерфейс и сервер.

2.2 Добавление пользовательской формы для внесения правок

В качестве Markdown редактора был выбран SimpleMDE, так как он уже применялся в интерфейсе администратора для тех же целей. Редактор доступен только авторизованным пользователям, которые нажали на соответствующую кнопку. Если же пользователь не авторизован, то нужно показать форму входа, как на рисунке 2.1.

Также требуется реализовать возможность выбор языка локализации.

Примерный интерфейс пользователя представлен на рисунке 2.2.

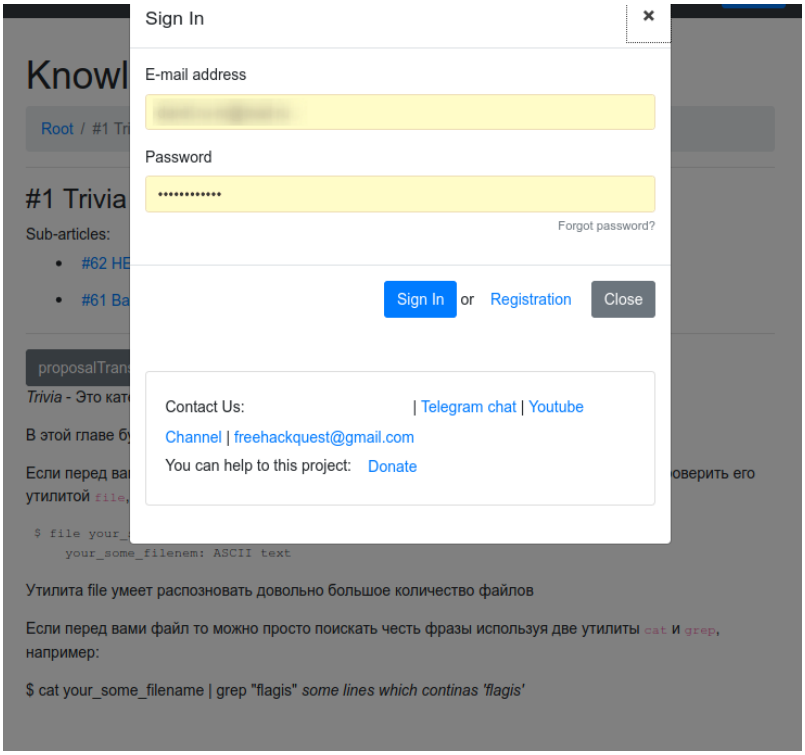

Рисунок 2.1 – Форма входа

По нажатию кнопки «Отправить» на сервер уходит запрос типа

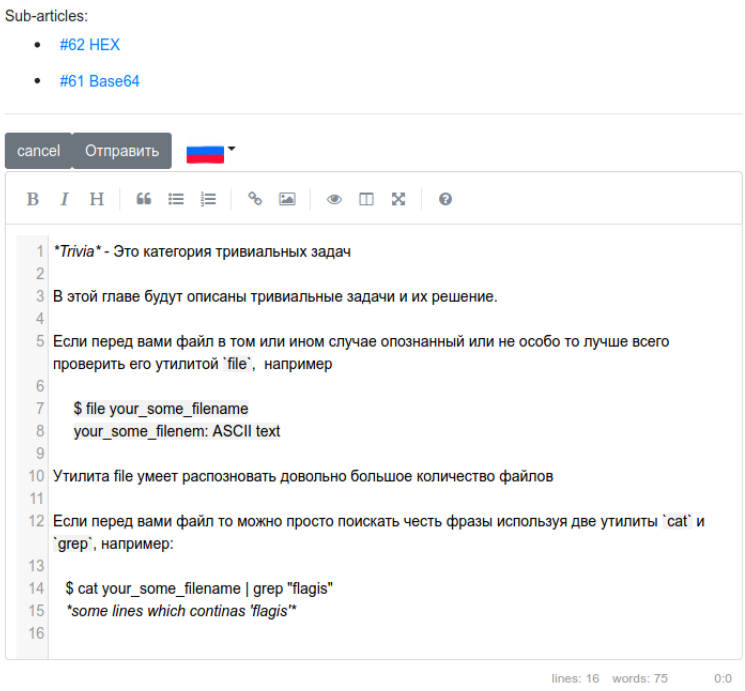

Рисунок 2.2 – Интерфейс редактора

classbook_proposal_add_record, в котором содержится предлагаемое содержимое, язык, идентификатор статьи.

Если добавление правки в базу данных прошло успешно, то на веб-интерфейсе отобразится сообщение, показанное на рисунке 2.3

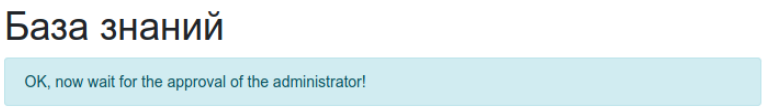

Рисунок 2.3 – Сообщение пользователю

2.3 Принятие и отклонение правок пользователей администратором

Что касается интерфейса администратора, то нужно было добавить отображение списка правок для данной статьи с отображением языка, возможностью редактировать, принять или отклонить правку. Причём кнопка Approve должна отображаться только для предложений.

Новый интерфейс администратора представлен на рисунке 2.4.

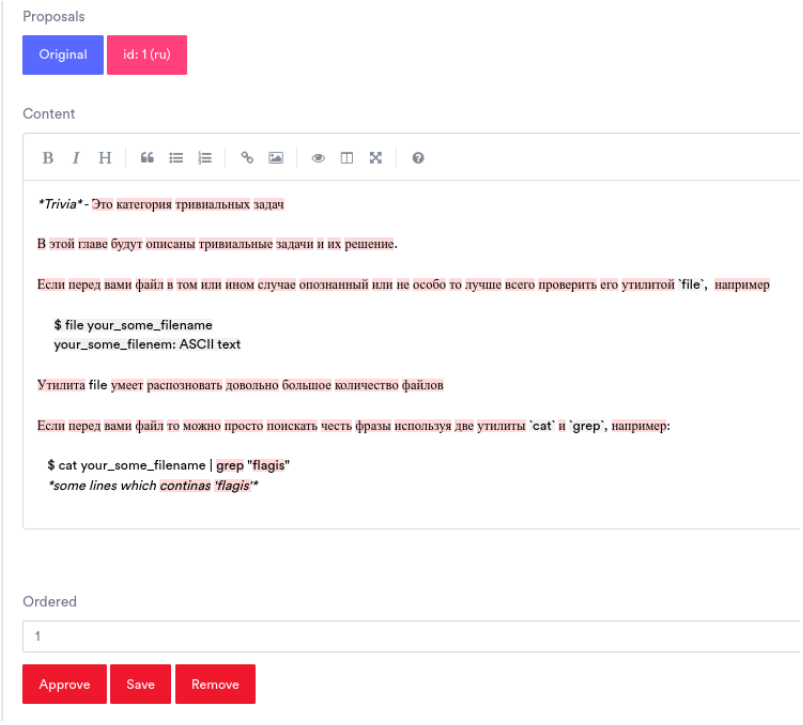

Рисунок 2.4 – Интерфейс администратора

Текущий выбор варианта статьи (оригинал или правка) для редактирования отображается другим цветом (розовым в данном случае) в списке «Propasals».

Данный список подгружает с сервера запросом classbook_proposal_list, который принимает идентификатор статьи.

Кнопка «Approve» отвечает за принятие правки, с помощью запроса classbook_proposal_approve.

Кнопка «Save» обновляет предложение или оригинальную статью, с помощью запроса classbook_proposal_update или classbook_update_record соответственно.

Кнопка «Remove» удаляет предложение или статью вообще, с помощью запроса classbook_proposal_remove или classbook_remove_record соответственно.

2.4 Серверная часть

Для функционала, описанного выше, были добавлены 2 новых типа запроса:
classbook_proposal_approve и classbook_proposal_update. Оба принимают идентификатор правки, а второе ещё и новое содержимое.

Структура базы данных, охватывающая 3 таблицы: classbook, classbook_proposal, classbook_localization, представлена на рисунке 2.5.

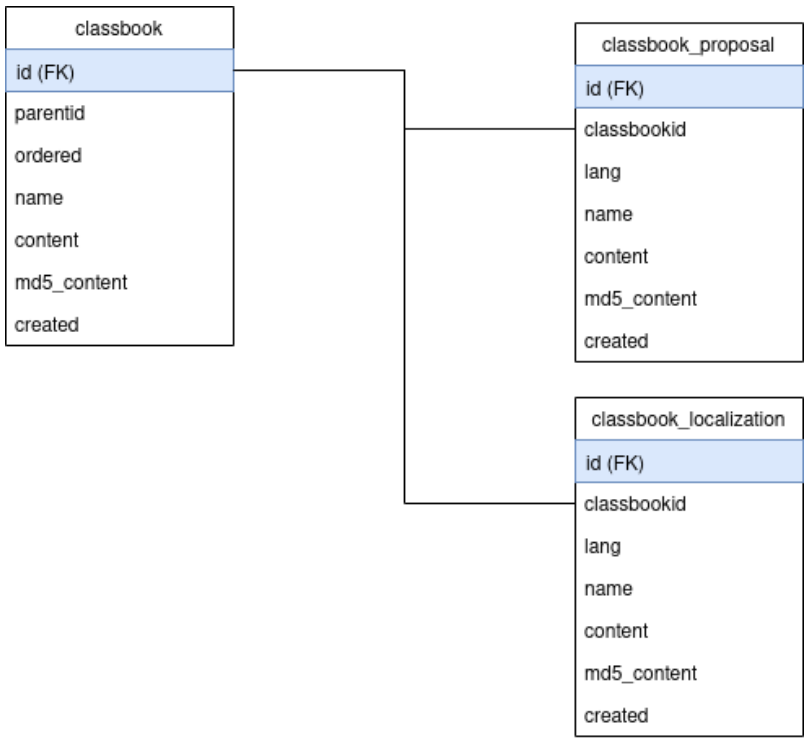

Рисунок 2.5 – Структура базы данных

В classbook хранятся метаинформация о статье (родительская статья, позиция, дата создания, md5 от контента) и само содержимое (имя, контент).

В classbook_localization хранятся данные локализации(язык, имя, контент) и отношения к оригинальной статье (идентификатор статьи).

В classbook_proposal хранятся правка (язык, имя, контент) и само содержимое (имя, контент).

В classbook_proposal_approve нужно получить идентификатор правки и найти саму статью, код на рисунке 2.6 реализует это.

```
nlohmann::json jsonRequest = pRequest->jsonRequest();

int nClassbookProposalID = -1;
if (jsonRequest.find("classbook_proposal_id") != jsonRequest.end()) {
    nClassbookProposalID = jsonRequest.at("classbook_proposal_id").get<int>();
}

if (nClassbookProposalID == -1) {
    pRequest->sendMessageError(cmd(), WSJCppError(404, "This proposal doesn't exist"));
    return;
}

QSqlDatabase db = *(pDatabase->database());
QSqlQuery query(db);

query.prepare("SELECT classbookid, content FROM classbook_proposal WHERE id = :classbook_proposal_id");
query.bindValue(":classbook_proposal_id", nClassbookProposalID);
if (!query.exec()) {
    pRequest->sendMessageError(cmd(), WSJCppError(500, query.lastError().text().toStdString()));
    return;
}
if (!query.next()) {
    pRequest->sendMessageError(cmd(), WSJCppError(404, "This proposal doesn't exist"));
    return;
}
```

Рисунок 2.6 – Получение идентификатора статьи

В зависимости от языка происходит выбор в какую таблицу производить запись, код на рисунке 2.7 производит запись в таблицу classbook, если язык правки - русский:

```
if (sLang = "ru") {

    query.prepare("UPDATE classbook SET content = :content, md5_content = :md5_content WHERE id = :classbookid");
    query.bindValue(":classbookid", nClassbookID);
    query.bindValue(":content", QString::fromStdString(sContent));
    query.bindValue(":md5_content", QString::fromStdString(sContentMd5_));
    if (!query.exec()) {
        pRequest->sendMessageError(cmd(), WSJCppError(500, query.lastError().text().toStdString()));
        return;
    }

    nlohmann::json jsonResponse;
    pRequest->sendMessageSuccess(cmd(), jsonResponse);

    return;
}
```

Рисунок 2.7 – Обновление статьи

Обновление правки в запросе classbook_proposal_update происходит в 2 этапа:

– Валидация входных данных - не отличается от соответствующего кода в предыдущем запросе.

– Запись в БД, которая представлена на рисунке 2.8

```
query.prepare("UPDATE classbook_proposal SET content = :content, md5_content = :md5_content WHERE id = :classbook_proposal_id");
query.bindValue(":classbook_proposal_id", nClassbookProposalID);
query.bindValue(":content", QString::fromStdString(sContent));
query.bindValue(":md5_content", QString::fromStdString(sContentMd5_));
if (!query.exec()) {
    pRequest->sendMessageError(cmd(), WSJCppError(500, query.lastError().text().toStdString()));
    return;
}

nlohmann::json jsonResponse;
pRequest->sendMessageSuccess(cmd(), jsonResponse);
```

Рисунок 2.8 – Обновление правки