МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Адаптивний алгоритм розрахунку часових проміжків для автоматизованої системи тестувань студентів»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Берест О.Б.**

Студентка групи ІНм – 92 — **Литюга О.Є.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2020 р.

ЗАВДАННЯ
до випускної роботи

Студентки четвертого курсу, групи ІНм-92 спеціальності "Інформатика" денної форми навчання Литюги Ольги Євгенівни.

Тема: " **Адаптивний алгоритм розрахунку часових проміжків для автоматизованої системи тестувань студентів**"

Затверджена наказом по СумДУ
№_____ от _____2020 р.

Зміст пояснювальної записки: 1) Інформаційний огляд і постановка задачі; 2) Вибір методів вирішення поставленої задачі; 3) Реалізація програмного засобу.

Дата видачі завдання "_____"_____2020 р.
Керівник випускної роботи _____ Берест О. Б.
Завдання прийняла до виконання _____ Литюга О. Є.

РЕФЕРАТ

Записка: 95 стор., 23 рис., 4 табл., 1 додаток, 20 джерел.

Об’єкт дослідження — процес проектування адаптивного алгоритму розрахунку часових проміжків для автиматизованої системи тестування, проектування баз даних, робота з Телеграм API

Мета роботи — розробка адаптивного алгоритму падбору часу для автоматизованої системи тестування для студентів на базі месенджеру Телеграм

Методи дослідження — методи проектування баз даних, методи проектування автоматизованих систем, методи розробки алгоритмів.

Результати — реалізований адаптивний алгоритм розрахунку часових проміжків**,** система тестування на базі меседжеру Телеграм. Була освоєна робота з JDBC модулем для роботи з базою даних, було використано Телеграм Bot Api та застосовано org.telegram.telegrambots бібліотеку для розробки. Система задовольняє усім вимогам, вказаним в постановці задачі. Додатково роботі була спроектована стуктура бази данних, яка дозволяє зберігати і маніпулювати данними потрібними для реалізації алгоритму і системи тестування

АЛГОРИТМ, ТЕЛЕГРАМ, БОТ, JAVA, СЕРЕДОВИЩЕ РОЗРОБКИ, БАЗА ДАНИХ, ЗАПИТ

ЗМІСТ

ВСТУП

Платформа Телеграм[1] - яілфється некомерційним проектом що має також відкритий вихідний код. Телеграм кросплатформений, тому його можна відкрити на декількох різних девайсах Особливістью месенджеру є наявність Bot API[3], що може реалізовувати логіку для написання ботів.

Боти[2] - спеціальні акаунти в Telegram, створені для того, щоб автоматично обробляти та відправляти повідомлення. Звичайні користувачі маєть можливість взаємодіяти з аккаунтами ботів за допомогою миттєвих повідомлень, що відправляються через групові або звичайні чати. Логіка бота контролюється за допомогою HTTPS запитів до нашого API для пошукових роботів. Особливість роботи бота - принцип запит-відповідь, бот реагує на надіслане йому повідомлення, обробляє його та надсилає йому відповідь.

В ході роботи буде реалізовано адаптивний алгоритм розрахунку часових проміжків для автоматизованої системи тестувань студентів на основі інтерфейсу бота. Алгоритм має адаптуватися до швидкості проходження тесту різними студентами, вірності виконання завдання.

Додаток буде використовуватися для проведення тестування студентів на підготовчих курсах та у навчальному процесі. За допомогою алгоритму студенти будуть виконувати завдання у оптимальний час для уникання надмірного споживання часу.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Поняття СУБД. Моделі баз даних. Реляційна модель

Система керування базами даних[20] (СУБД) [4] — набір взаємопов'язаних даних, які називаються базою даних, а також набір програм для можливого доступу до цих даних. Надає можливості оновлення, створення, пошуку і збереження даних у базах даних із контролем доступу до збереженої інформації.

Можливості СУБД:

1. Є можливість створення БД
2. Має надавали доступ до операцій вставки, видалення, оновлення та читання інформації з БД
3. Є можливість надавати користувачам контрольований доступ до БД

Модель бази даних[5] - тип моделі даних, яка визначає логічну структуру бази даних і принципово визначає, яким чином дані можуть бути збережені, організовані і оброблені. Найбільш популярним прикладом моделі бази даних є реляційна модель, яка використовує табличний формат.

У світі технологій баз даних існує два основних напрямки[6]: SQL і NoSQL, реляційні та нереляційні бази даних. Відмінності між ними полягають в тому, як вони спроектовані, які типи даних підтримують, як зберігають інформацію.

1. Реляційні БД[7] зберігають структуровані дані, які зазвичай представляють об'єкти реального світу. Це можуть бути, скажімо, короткі відомості про людину, або про товари у кошику в магазині, що згруповані в таблицях, формат товарів, в той час, заданий уже на етапі проектування сховища.

Причини, які можуть послужити приводом для вибору SQL-бази:

1. Необхідність виконувати вимоги ACID, а саме - атомарність, несуперечливість, ізольованість та довговічність. Це допомагає зменшити ймовірність непередбачуваної поведінки системи і забезпечити цілісність даних у базі.

2. Основна структура даних не буде змінюватись, а дані можливо структурувати.

SQL-бази можуть бути застосовані в проетах такого типу:

1. Вимоги до даних можливо знати заздалегідь.

2. Технологія повинна бути надійна і добре рекомендована. Щоб можна було розраховувати на досвіт освічених розробників, а недоліки були відомі попередньо.

3. Цілісність даних пріорітетна.

1.2. Опис СУБД MariaDB

MariaDB[20] фактично - це відгалуження від СУБД MySQL, що розробляється спільнотою під ліцензією GNU GPL. Розробка та підтримка сайта MariaDB здійснює компанія MariaDB Corporation Ab і фонд MariaDB Foundation. Поштовхом до створення стала необхідність забезпечення вільного статусу СУБД, на противагу політиці ліцензування MySQL компанією Oracle. Система ліцензування MariaDB зобов'язує учасників, бажаючих додати свій код в основну гілку СУБД, обмінюватися своїми авторськими правами з MariaDB Foundation для охорони ліцензії і можливості створювати критичні виправлення для MySQL.

Вона повністю сумісна з MySQL, і прекрасно підходить в якості заміни, тому що повністю відповідає як набір команд, так і API.

Особливості:

1. Система працює швидко.

2. Індикатори дадуть вам знати, як обробляється запит.

3. Розширювана архітектура і плагіни дозволяють налаштовувати інструмент відповідно до ваших потреб.

4. Шифрування є в мережі, сервері і на рівні додатку.

5. Рушій досить новий, тому поки немає ніяких гарантій подальших оновлень.

6. Як і в багатьох інших безкоштовних базах даних, вам доведеться платити за підтримку.

7. Ідеальна як альтернатива MySQL, якщо MySQL не влаштовує з якихось причин.

MariaDB являється відгалудженням СУБД Mysql, яка має в перевазі ті ж характеристики і особливості, але, ліцензію на Mysql викупила Oracle[8] і частково випускається згідно своїх ліцензій, натомість MariaDB – повністю безкоштовна СУБД з відкритим вихідним кодом. І, як нове відгалудження від Mysql[9], MariaDB[10] частіше випускає свої оновлення. MariaDB зараз стрімко розвивається і має більше можливостей. Ці можливості стосуються оптимізації, поліпшення роботи з пам'яттю, і багато іншого. Згідно попереднього, MariaDB[11] являється потрібною СУБД для даного проекту. Також, можна додати, що її подальшу реалізацію схеми на діалекті SQL можна з легкістю перенести на MySQL СУБД, адже синтаксис залишається тим же. Тому орієнтованою СУБД для розробки являється MariaDB[12]. Також MariaDB, як і MySQL успішно інтегруються з мовою програмування Java. Для MariaDB присутній JDBC driver, що робить операції з бд із програми безпечними та надійними.

1.3. Опис основних мов програмування

C++ – це мова програмування високого рівня[20], яка підтримує певні парадигми програмування: узагальненої, об'єктно-орієнтованої та процедурної. Мова як правило застосовується для системного програмування, написання драйверів, високопродуктивних програм, що будуть виконуватись на сервері або ж клієнтському комп’ютері.

C# – мова програмування, яка являється об'єктно-орієнтованою, з безпечною системою типізації, розроблена для платформи .NET. Перевагою даної мови є те, що розмір вихідних програмних продуктів є відносно невеликим і не потребує додаткових бібліотек для запуску. Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, перевантаження операторів, вказівники на функції-члени класів, події, властивості, винятки, коментарі у форматі XML. Основуючись на своїх попередниках — мовах C++, Delphi і Smalltalk — C# виключає деякі потенційно нестабільні лоігчні моделі, як множинне спадкування класів.

Java[13] – також об'єктно-орієнтована мова програмування високого рівня, утім вона проектувалась саме у такій парадигмі від самого свого створення. В цьому вона вигідно відрізняється від C++, який, з міркувань сумісності, має багато успадкованих елементів процедурної мови C. Java[14] у великій мірі запозичила синтаксис із C і C++, наприклад, об'єктну модель було перейнято із C++, хоча і допрацьовано. Архітектурно була усунута можливість появи ряду конфліктних ситуацій та було спрощено сам процес розробки програм об'єктно-орієнтованих[15]. Ряд дій, які в C/C++ мають бути виконані програмістом, доручено в java віртуальній машині JVM[16].

1.4. Поняття алгоритму. Нейронних мереж. IEI-технологія

Алгоритм – це скінченна послідовність чітко визначених, реалізованих на комп'ютері інструкцій, як правило, для вирішення класу задач або для

обчислення. Алгоритми завжди однозначні і використовуються як специфікації для виконання розрахунків, обробки даних, автоматизованих міркувань та інших завдань.

Нейронні мережі[18] (NN), - це обчислювальні системи, які нечітко натхненні біологічними нейронними мережами, що складають мозок тварин. Глибоке навчання є частиною більш широкого сімейства методів машинного навчання, заснованих на штучних нейронних мережах з репрезентативним навчанням. Глибоке навчання дозволяє навчати модель передбачати результат по набору вхідних даних. Головною одиницею являється нейрон, Нейрони згруповані в три різних типи шарів: вхідний шар; прихований шар (шари); вихідний шар. випадку ми не матимемо стільки даних, яких система потребує для навчання. IEI-технологія[19] полягає у трансформації апріорного в загальному випадку нечіткого розбиття простору ознак до чіткого розбиття класів еквівалентності за допомогою ітераційною оптимізацією параметрів функціонування інформаційної системи. Основна ідея навчання за IEI-технологією полягає в послідовній нормалізації вхідного математичного опису ІС шляхом цілеспрямованої трансформації апріорних габаритів розкиду реалізацій образів з метою максимального їх захоплення контейнерами відповідних класів, що відбудовуються в радіальному базисі у процесі навчання.

1.5. Постановка задачі

Автоматизована система тестування реалізована на мові Java, інформаційна система спроектована в СУБД MariaDB. При проходженні тестування студенту дається певний час на виконання завдання (за замовчуванням – 5 хвилин), надалі, з проходженням тесту іншими студентами, система повинна адаптувати час, який надається для виконання завдання, адаптувати згідно до попередніх результатів, тобто за який час студенті відповідали або чи відповідали вірно.

Основна логіка описується як: запуском куратором тесту для групи, а потім проходження кожним студентом окремо тесту, через діалог з ботом. Паралельно куратор може переглядати результати проходження тестування групою. Розглянути вимоги можна детальніше:

1. Користувач в процесі реєстрації повинен вказати своє ім’я

2. Адмін створює групу для студентів і назначає кураторів для цієї групи.

3. У куратора є можливість створити тест для студентів, побачити список своїх створених тестів, починати тест для групи і побачити як студенти пройшли тест з балами.

4. Після початку тесту куратором, студент групи має отримати повідомлення про початок тесту, далі він відповідає на всі запитання і отримує від системи резутат з оцінкою у процентному вигляді.

5. Студент має обмеження по часу для виконання завдання, при отриманні завдання він знає скільки часу відведено на завдання.

6. Якщо студент не встиг виконати завдання вчасно, завдання вважається незданим і студент переходить до іншого завдання.

2. ВИБІР МЕТОДІВ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1. Проектування бази даних

Логіка додатку базується на вводі даних від користувачів, обробкою їх системою паралельно записуючи проміжні дані до сховища та відповіді результату користувачу, деталі див на DFD діаграмі.

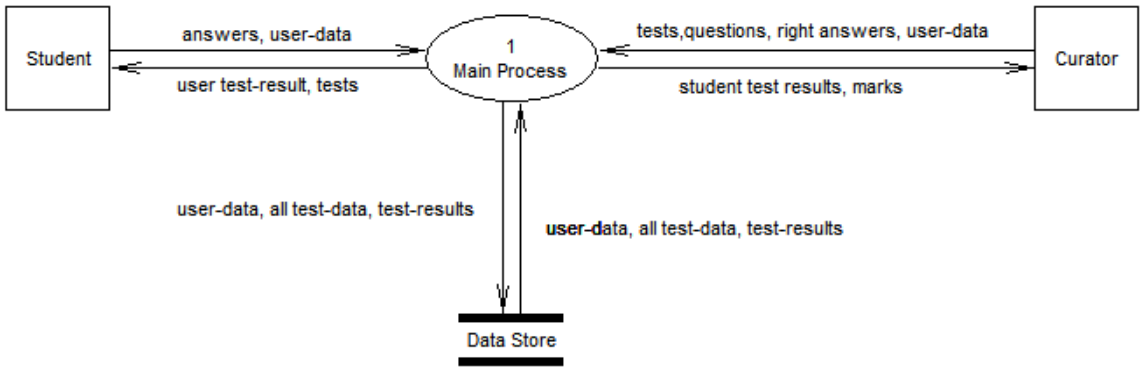


Рисунок 2.1 – DFD діаграма

Згідно до вимог, що вказані у пункті 1.5 мною була створена ERD діаграма рис 2.2,

user_id - унікальний ідентифікатор користувачів, що присвоюється платформою Телеграм. Основною таблицею в базі являється сутність users (користувачі), що зберігає дані про користувача: user_id, ім’я, нік, поле для ролі: моодератор/адмін та додаткова інформація.

states - важлива сутність, що зберігає стан юзера за user_id. В таблиці зберігаються дані про номер тесту, запитання. На дані в цій таблиці додаток опирається, щоб запустити подальшу логіку для користувача.

Groups/ users_groups – сутності, що зберігають в собі дані про групи і дані про зв’язок користувачів з групами та дані про ролі користувача в цій групі (студент або куратор).

Tests/ questions – сутності, що мають в собі дані про тести: їх назву, який користувач створив, а для запитання: до якого тесту прив’язане питання, тип запитання, яка правильна відповідь, час, що відведено на запитання.

answers - сутність у якій кожен запис містить у собі відповідь на певне запитання, що ідентифікується за question_id. Також присутні дані продату відправлення запитання, дату завершення (або ні) ,за який час виконав завдання студен і чи вірно він його виконав, також id поточного тесту, дані про який можна переглянути у таблиці done_test. У цій сутності хберігається інформація про будь-які тести, зо колись були розпочаті. У сутності присутні такі поля: id поточного тесту, оцінка, дата старту та завершення тестування для користувача, пройдений тест чи ні (виставляється по завершенню тесту)

Рисунок 2.2 – ER діаграма

2.2. Вибір середовища розробки

IntelliJ IDEA Ultimate Edition компанії JetBrains[17] було обрано як середовище для розробки додатку. З огляду на численні рекомедації програмістів, засоби налогодження(Debug – режим), оптимальний графічний інтерфейс, зручна інтеграція з Maven, що автоматично підкачує усі залежності та бібліотеки, легке підключення до бази даних напряму з середовища, інтеграція з системою конролю версій, що дозволяє виконувати комміти напряму з середовища. Наявність позитивного досвіду роботи з WebStorm та Commuity версією IntelliJ IDEA також від JetBrains вплинуло на мій вибір, так як продукти

мають схожий інтерфейс, що дозволяє мені швидше адаптуватися до роботи в цьому середовищі. На рис. 2.3 можна побачити приклад графічного інтерфейсу середовища з брейкпоінтами для дебаг режиму.

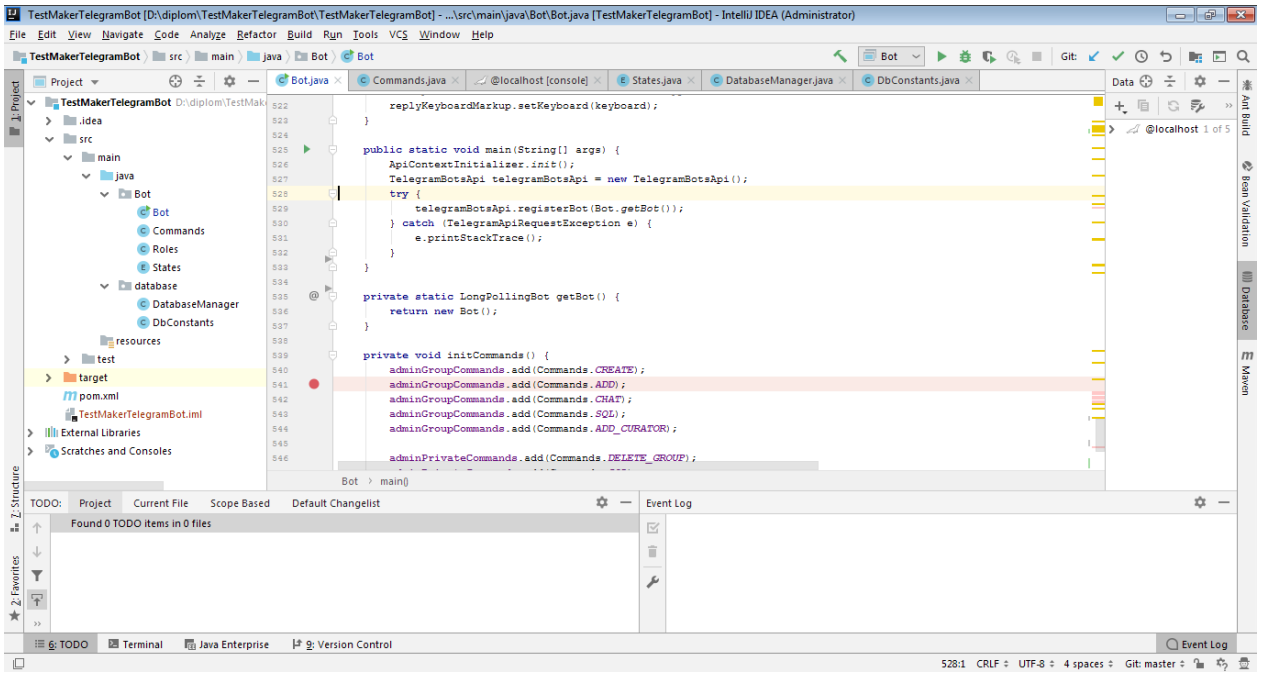


Рисунок 2.3 - Зовнішній вигляд середовища розробки

2.3. Визначення команд управління системою

Для даного додатку[20] можна визначити 4 основні ролі, які може мати користувач і один користувач може мати декілька ролей. Цими ролями є Звичайний користувач, Адміністратор, Студент групи та Куратор групи. Можливості цих ролей наявно показано на діаграмі можливостей на рисунку 2.4

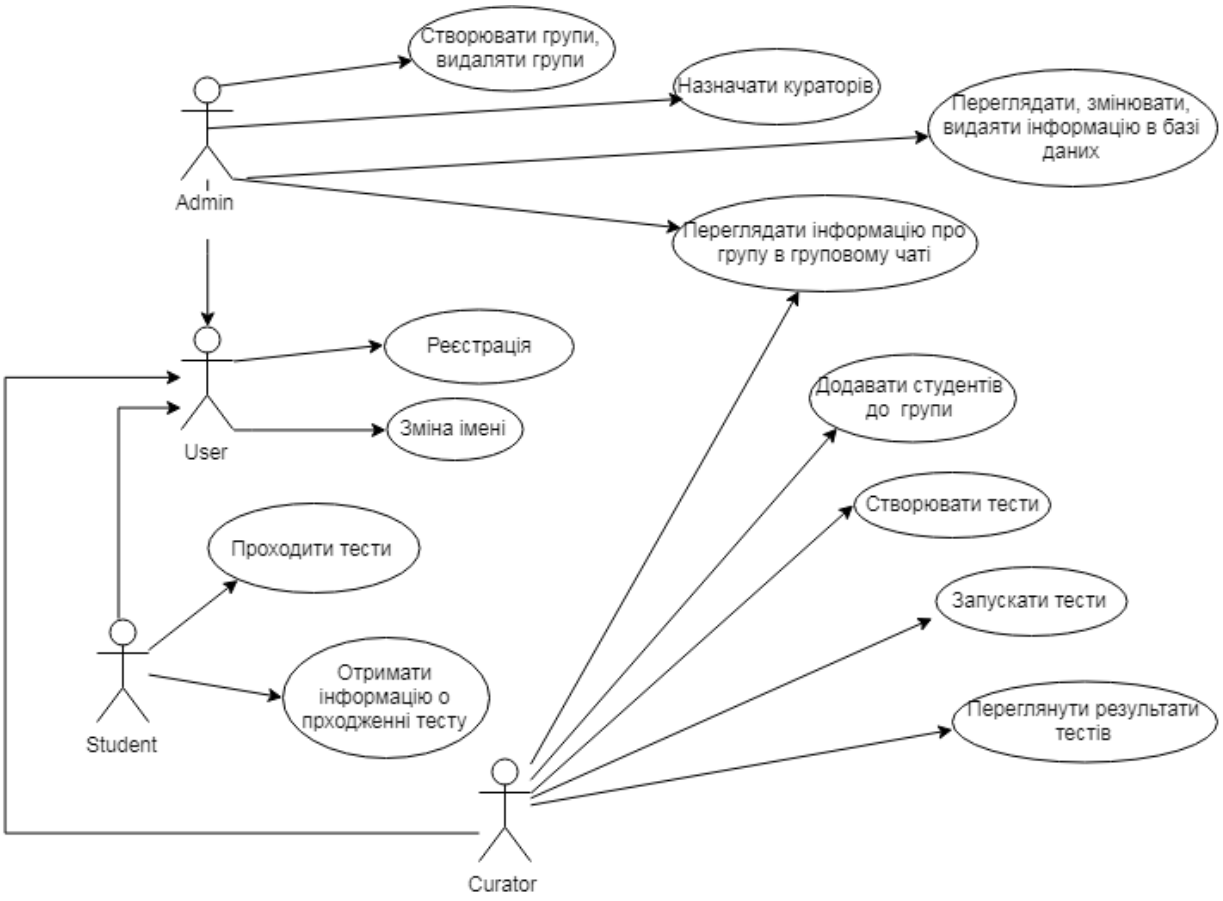


Рисунок 2.4 - Діаграма можливостей

Команди для телеграм-боту зазвичай починаються зі знака слешу “/” англійською мовою. Визначимо список команд для кожної дії користувача, які вказані на рисунку 2.4.

/change_name – команда для зміни імені, введеного при реєстрації, при введені імені довшого за 200 символів, система його обріже до потрібного розміру

/sql – команда лише для Адміністратора, наприклад /sql select * from users – покаже усі дані про користувачів, зареєстрованих у системі

/create – для створення групи, наприклад /create group2a

/delete_group – для видалення групи для даного групового чату

/chat – для переглядання інформації про групу

/add_curator – назначає куратора групи, використовується відповіддю(reply) на повідомлення потрібного користувача

/add – додає користувача студентом до групи, використовується аналогічно команді /add_curator

/create_test – доступна лише користувачам, які являються кураторами хоч однієї групи, при натисканні на неї створюється тест і надається можливість додавати до нього запитання

/add_question– для додавання запитання у тест, id – test_id потріббного тесту, команда генерується самостійно системою і користувачу непотрібно прописувати її самостійно

/show_tests – команда, яка виводить усі тести, створені куратором і генерує команду /start_test для кожного тесту

/start_test_[id] – для початку тесту усіх студентів заданої групи, id – test_id потріббного тесту, аналогічно генерується ботом командою /show_tests

/show_results_[id] – де id test_id потріббного тесту,генерується командою /start_test і виводить результати усіх студентів даної групи, які проходять/пройшли даний тест

2.4. Визначення основних методів програмування додатку

Коротко основи роботи телеграм-ботів[20] можна пояснити принципом запит-відповідь, але так як нам потрібно сформувати більш складну систему зі зберіганням даних, на рисунку 2.5 можна переглянути основну схему роботи системи.

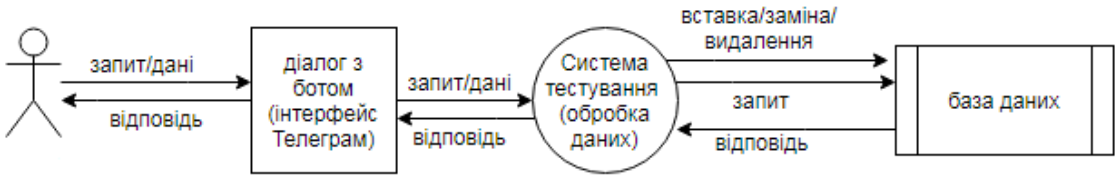


Рисунок 2.5 - Принцип роботи додатку

Основною особливістю від звичайних десктоп або веб-додатків являється неможливість крок-за-кроком логіки, телеграм Апі не зберігає сесію користувача, не можна запрограмувати поступово запитання-відповідь послідовність. Системі просто приходить Update об'єкт який зберігає у загальну собі інформацію про користувача та повідомлення, яке він надіслав. Тому у пункті 2.4 був створений список команд, які дозволяють системі дізнатися, яку інформацію потребує користувач, але такі дії, як ввести своє ім’я, надіслати відповідь на запитання важко оформити в мову команд і це буде незручно для користувачів, тому для даного додатку реалізований принцип збереження станів. Для збереження стану буде використовуватися база даних і по занесеному стану у базі система зрозуміє, як саме обробляти подані до неї дані. Також, так як специфіка додатку полягає у проходженні тестів, для користувача, який проходить тест буде зберігатися унікальній ідентифікатор тесту (id) та запитання.

2.5. Адаптивний метод підбору часу

Тестування студентів в режимі онлайн потребує певних обмежень у часі, які задає викладач. Проте ми стикаємося з проблемою, що людина не може коректно оцінити час, який потрібен студенту для виконання завдання: деякі можуть бути занадто складні, а інші навпаки вирішуватимуться миттєво.

Для даного додатка постала задача: вибрати і реалізувати певний алгоритм, який на основі того, як завдання вирішували інші студенти, підбере оптимальний час для виконання завдання для наступного студента.

В п.1.4 було описано декілька методів, які можна використати для вирішення задачі. Якщо ми візьмемо глибоке навчання, то вхідним шаром будуть дані про те як студенти виконали завдання, а на виході ми маємо отримати найбільш оптимальний час. Для масштабної програми, де тести проходять мільйони користувачів цей метод був би оптимальним, але в нашому випадку ми не матимемо стільки даних, яких система потребує для навчання. Також ми розглянули IEI-технологію.

Проаналізувавши сет вхідних даних та специфіку застосування(нам не потрібна велика точність), було додатково розглянуто декілька способів для реалізації поставленої задачі. Було вирішено написати власний більш простий алгоритм.

Так як програмно є можливість задавати час, за який студент може виконати завдання, то ми маємо такий вхідний сет для одного завдання: стартовий час(час, який був наданий студенту на виконання завдання); час, за який студент виконав завдання; чи встиг дати відповідь взагалі; чи вірно відповів на запитання. На виході ми повинні отримати як зміниться час, за який студент має відповісти на запитання див. рис. 2.6.

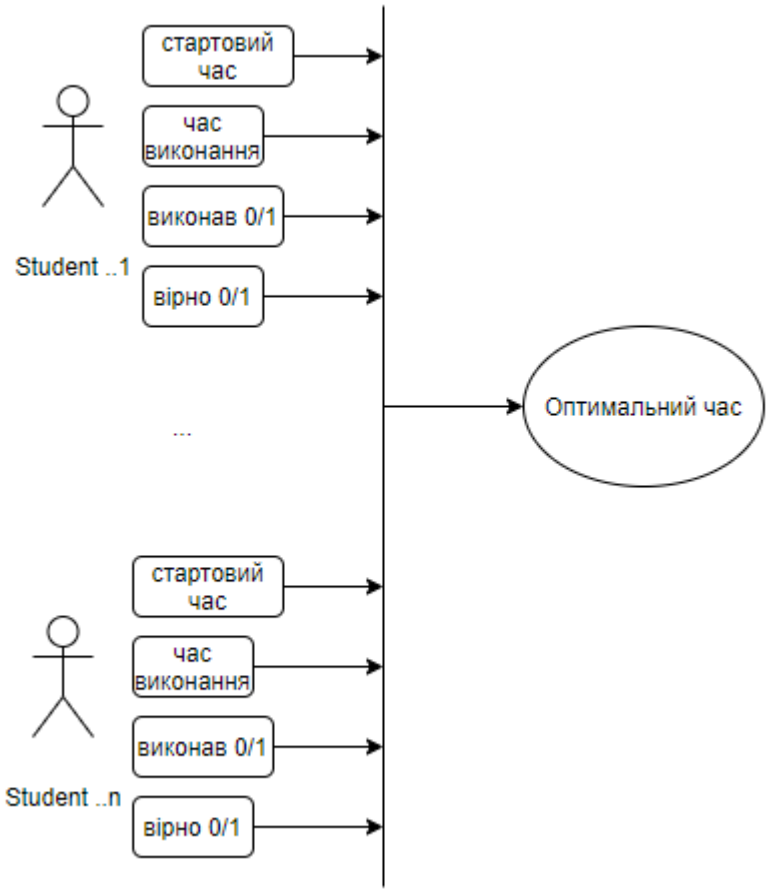


Рисунок 2.6 – Вхідні та вихідні дані

Спрощену схему алгоритму можна переглянути на рис. 2.7, де на вхід подаватиметься сет даних і в циклі буде проводитися обробка кожного студента і в кінці, на основі первинного аналізу, буде підраховуватися коефіцієнт, який потім помножиться на стартовий час і ми отримаємо в результат час, що нам потрібний.

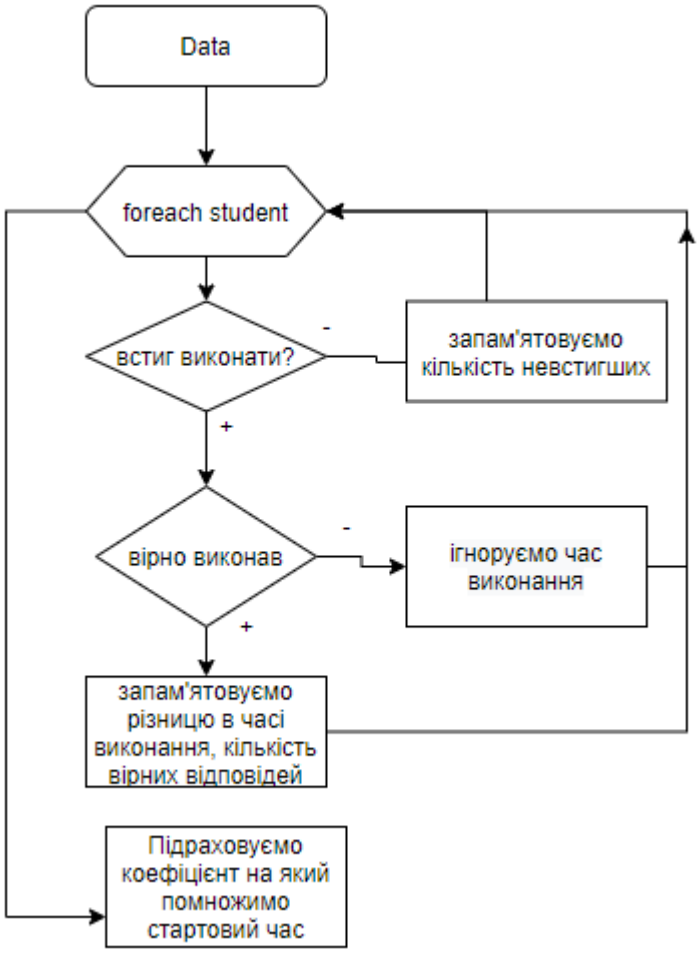


Рисунок 2.7 – Спрощена схема алгоритму

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Створення бази даних

Особливість MariaDB СУБД – синтаксис CREATE table IF NOT EXISTS, що буде використовуватися для створення таблиць для усунення помилок при повторному запуску скрипта.

Таблиці states та users можно переглянути в таблиці 3.1.

Таблиця 3.1 - Опис атрибутів сутностей users, states

Таблиця	Поле	Тип данних	Ключі	Обмеження
Users	user_id	INTEGER	PK	Не пустий
	name	CHAR (100)		
	nick	CHAR (30)		
	role	CHAR (30)		
	description	TEXT		
States	user_id	INTEGER	PFK - users	Не пустий
	state	CHAR2(100)		Не пустий
	cur_test_id	INTEGER	FK – answers	
	question_id	INTEGER	FK - answers	

id - унікальний ідентифікатир запису у сутностях tests , groups, done_tests, questions, який генерується автоматично за допомогою СУБД, синтаксично описується за допомогою лексеми AUTO_INCREMENT[12]. Поля цих таблиць можна побачити у таблиці 3.2.

Таблиця 3.2 - Опис атрибутів сутностей groups, tests, questions, done_tests

Таблиця	Поле	Тип данних	Ключі	Обмеження
groups	group_id	INTEGER	PK	Не пустий, auto_increment
	name	CHAR (100)		Унікальний
Tests	test_id	INTEGER	PFK - users	Не пустий
	name	CHAR(100)		Не пустий
	user_id	INTEGER	FK – answers	
questions	question_id	INTEGER	PK	Не пустий
	test_id	INTEGER	PFK - tests	Не пустий
	question	VARCHAR(2000)		
	type	CHAR2(330)		
	data	TEXT		
	answer	VARCHAR(2000)		
	filepath	CHAR2(200)		
	timer	INTEGER		
done_tests	cur_test_id	INTEGER	PK	Не пустий
	test_id	INTEGER	FK - users	Не пустий
	user_id	INTEGER	FK - tests	Не пустий
	start_time	DATE		
	end_time	DATE		
	mark	INTEGER		
	passed	INTEGER		

В таблиці questions для реалізації таймерів було створено поле timer, де зберігажться час, який дажться студенту на запитання.

Для таблиць answers , users_groups, унікальний ідентифікатор не задано, первинний ключ в цих сутностях – складовий, див. деталі у табл. 3.3.

Таблиця 3.3 - Опис атрибутів сутностей users_groups, answers

Таблиця	Поле	Тип данних	Ключі	Обмеження
users_groups	user_id	INTEGER	FK - users	Не пустий
	group_id	INTEGER	FK - groups	
	role	CHAR (100)		Не пустий
answers	user_id	INTEGER	FK - users	Не пустий
	is_right	INTEGER		
	cur_test_id	INTEGER	PFK – done_tests	Не пустий
	question_id	INTEGER	PFK - questions	Не пустий
	timer	INTEGER		
	start_time	DATETIME		
	end_time	DATETIME		
	answer	VARCHAR(2000)		

В таблиці answers start_time – заповнюється при відправленні запитання користувачу, як sysdate(), end_time - sysdate() при відповіді на запитання користувачем або, якщо користувач не встиг відповісти на запитання, то момент, коли час на відповідь сплинув.

3.2. Приклад типових запитів до бази даних

1. Вибір стану користувача за заданим user_id:

Select state from states where user_id = ?;

2. Вибір запитання і тесту, на якому в даний момент зупинився користувач:

Select cur_test_id, question_id where user_id = ?;

3. Вивести всіх студентів у групі:

Select user_id from users_groups where group_id = ? and role = ?;

Де параметр role - ідентифікатор ролі студента, наприклад “STUDENT”

4. Вибір усіх оцінок і інформації о сдачі/не сдачі усіх тестів користувача

Select mark, passed from done_tests where user_id = ?;

5. Вивести дані о усіх тестах даної групи

Select mark, passed from done_tests

where user_id in (

Select user_id from users_group

where group_id = ? and role = ?);

6. Вивести усі дані про запитання для певного тесту

```
select QUESTION_ID, TEST_ID,  QUESTION_ID, TYPE,
        DATA, ANSWER, FILEPATH, TIMER
 from QUESTIONS
where TEST_ID = ?
```

7. Запит для оновлення таймерів для запитань

update QUESTIONS set timer = ? where question_id = ?

8. Запит для обновлення даних при відповіді на запитання студентом

update ANSWERS set answer = ?, is_right = ?, end_time = sysdate(), timer = TIMESTAMPDIFF(SECOND, start_time, sysdate())

" where user_id = ? and cur_test_id = ? and question_id = ?

3.3. Програмна реалізація додатку

Код додатку розділяється на 2 пакети: модель та логіка боту.

Розглянемо model, package model.entities зберігає в собі основні сутності, якими маніпулює програма: користувачі, стани, тести, запитання, відповіді, виконані тести, можна докладніше переглянути на рис. 3.1.

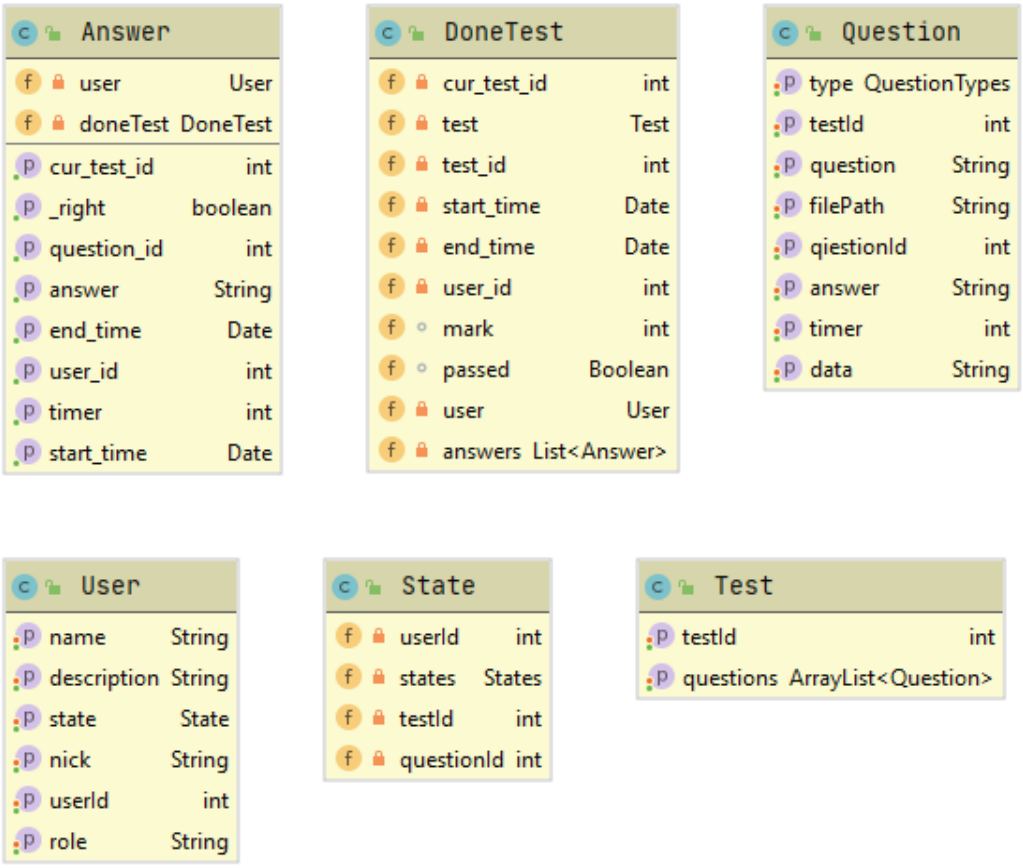


Рисунок 3.1 - Діаграма класів пакету model.entities

Розглянемо пакет model.database для роботи з базою даних, переглянути структуру пакету можливо на рисунку 3.2, змодельована даграма класів пакету.

Класс DoneTestAnswersDB включає в себе методи для маніпулювання сутностей: виконаний тест та відповіді, QuestionTestDB відповідно – завдання/тести, а UserStateDB для користувачів та їх станів.

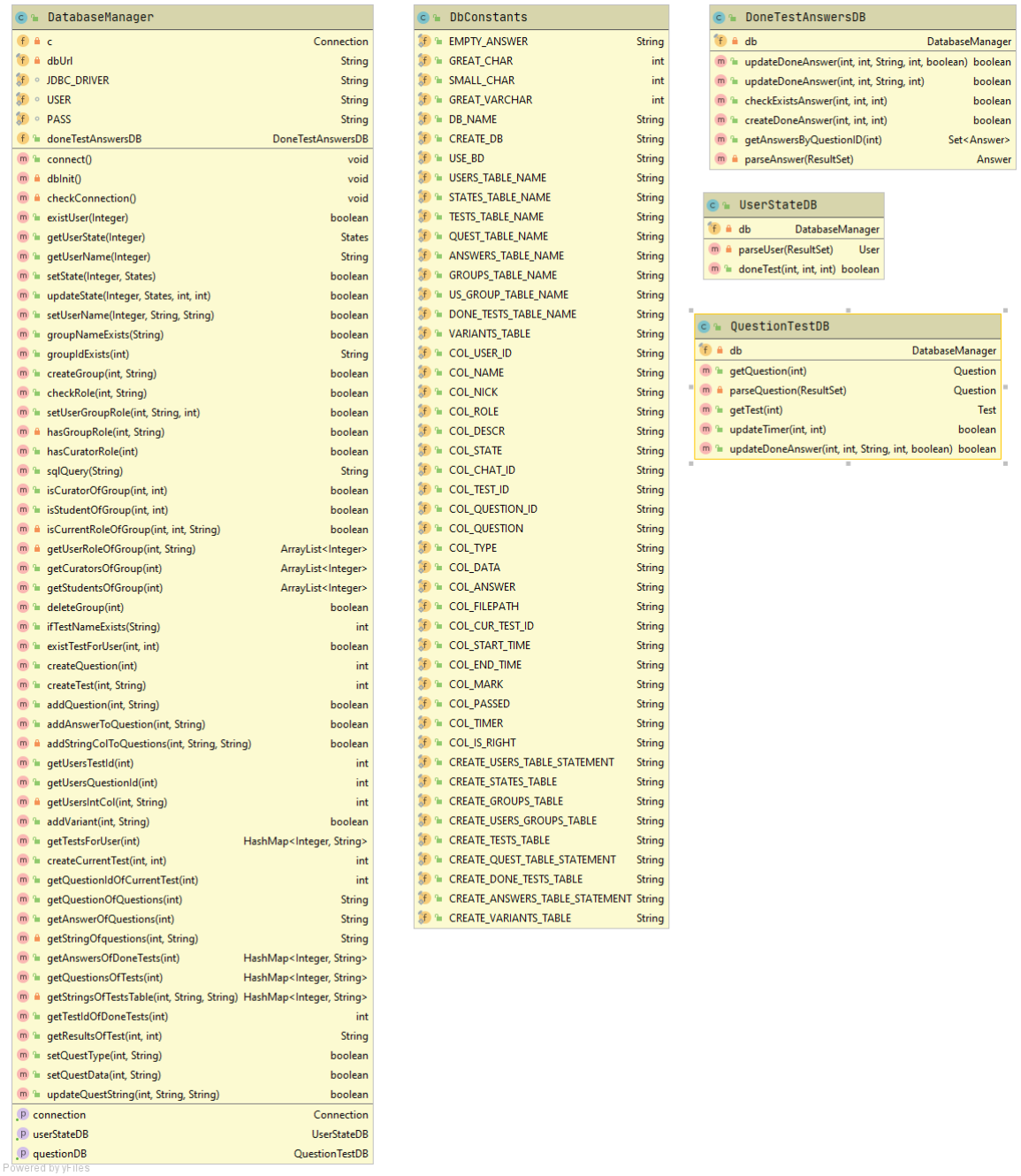


Рисунок 3.2 - Діаграма класів пакету model.database

Клас DbConstants зберігає у собі константи для моделювання запитів та роботою з базою даних

Константи GREAT_CHAR, GREAT_VARCHAR, SMALL_CHAR зберігають у собі обмеження на стовпці у таблицях відповідно 100, 2000, 30.

Константа DB_NAME –зберігає назву бази даних.

CREATE_DB – для створення бази даних з певним кодуванням.

USE_BD – для переходу на дану створену бд.

Константи, які закінчуються на TABLE_NAME несуть відповідне ім’я таблиці.

Константи, які починаються на COL_ - імена колонок у таблицях.

Константи, які починаються на CREATE_ - зберігають запит для створення відповідних таблиць, далі використовуються в DatabaseManager для ініціювання бази даних.

Клас DatabaseManager[20] містить у собі методи для роботи з базою даних.

Під час створення об’єкту класу в конструкторі ініціалізується змінна з рядком підключення до бд, в JDBC_DRIVER константі зберігається JDBC драйвер, в даному випадку це "org.mariadb.jdbc.Driver".

Основним методом являється void connect(), який генерує підключення дл бази даних, та визиває метод dbInit().

```
Class.forName(JDBC_DRIVER);
c = DriverManager.getConnection(dbUrl, USER, PASS);
dbInit();
```

Метод dbInit() ініціює базу даних, в ньому запускаються усі скрипти по створенню бази даних, переключення на цю базу даних та створення таблиць, якщо вони не існують.

У кожному методі класу, який маніпулює з базою даних, визивається метод checkConnection(), який перевіряє чи закрите підключення, та визиває метод connect(), якщо воно закрилось.

Для запитів був використаний PreparedStatement, який запобігає sql-ін’єкціям і дозволяє кешувати часті запити. Типовий запит:

String sql = "select " + COL_NAME + " from " + USERS_TABLE_NAME +

" where " + COL_USER_ID + " = ?" – де імена таблиці та стовпців взяті з класу DbConstants.

PreparedStatement preparedStatement = c.prepareStatement(sql);

preparedStatement.setInt(1, userId);

Для результату використовується ResultSet.

```
ResultSet resultSet = preparedStatement.executeQuery();
if (resultSet.next()) {
    name = resultSet.getString(1);
}
resultSet.close();
preparedStatement.close();
```

Розглянемо пакет model.keyboards для роботи з апі телеграму для створенні спеціальних клавіатур (завдання з кнопками) структуру пакету можливо на рисунку 3.3 на даграмі класів.

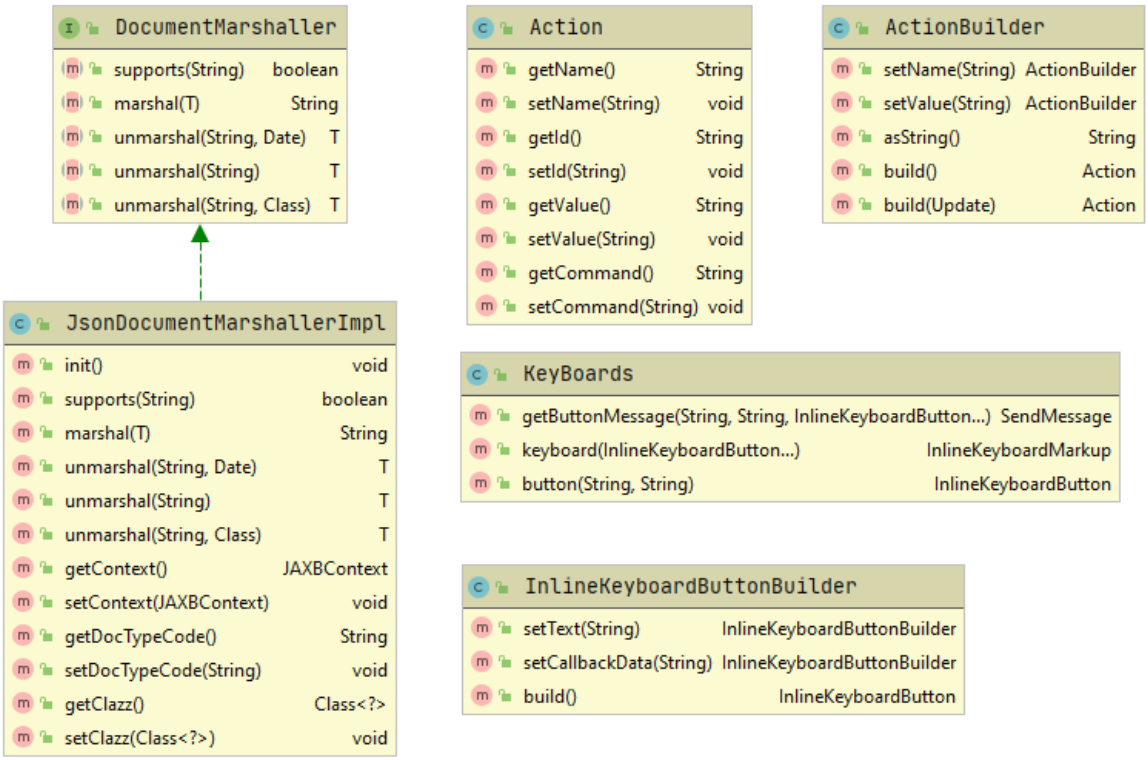


Рисунок 3.3 - Діаграма класів пакету model.keyboards

Розглянемо пакет Bot діаграму класів якого можна переглянути на рисунку 3.3.

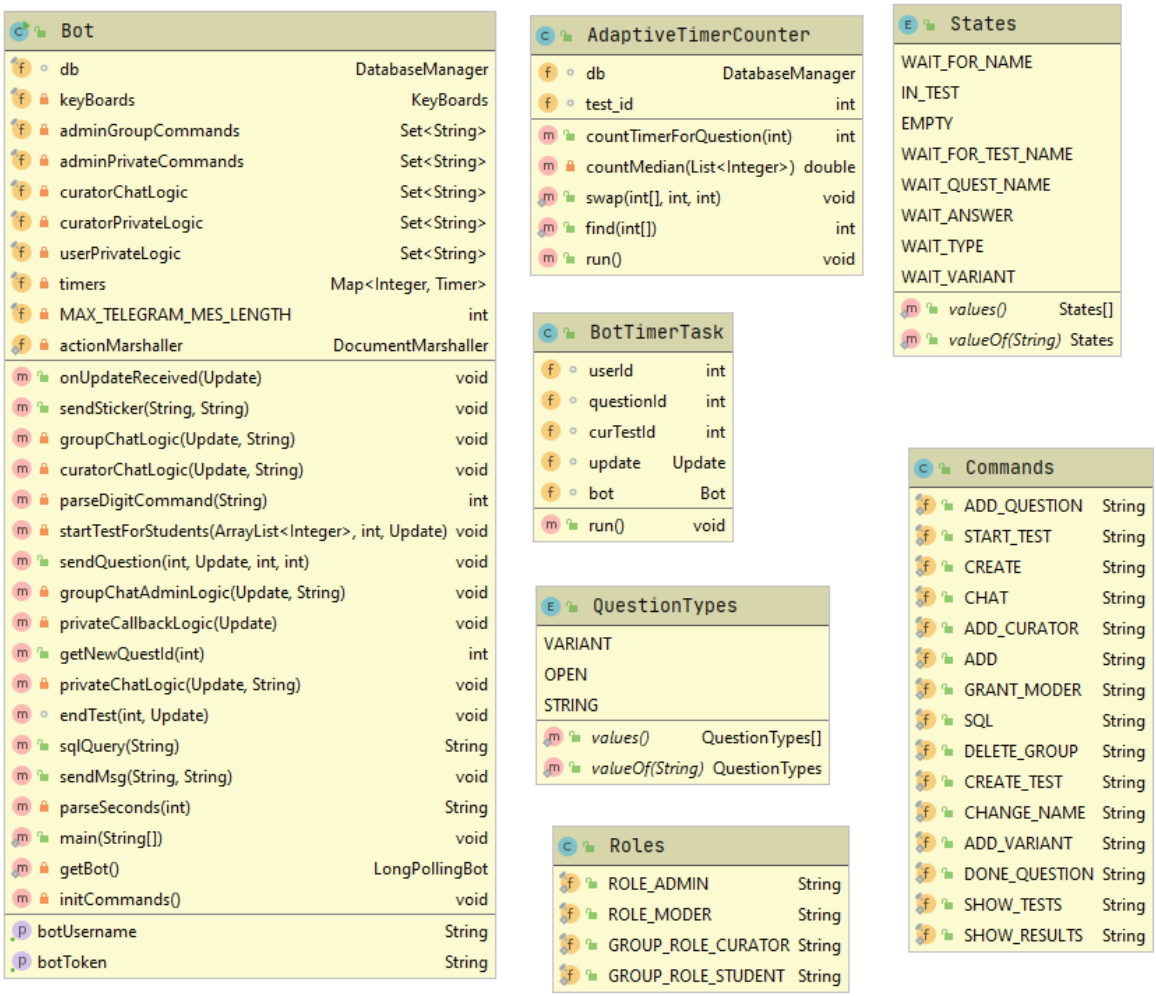


Рисунок 3.4 - Діаграма класів пакету Bot

Enum States створений для станів користувачів. Клас Commands зберігає константи з командами для бота, список команд ідентичний командам вказаним у пункті 2.4. Клас Roles містить у собі константи з основними ролями: роль адміністратора для користувача та ролі куратора і студента для групи.

Клас Bot наслідується від TelegramLongPollingBot, для цього потірбно перевизначити методи : getBotToken() - який повертає токен бота,

getBotUsername() – що сберігає юзернейм бота. Дані в цих двух методах задаються/видаються BotFather в Телеграм. Також потрібно перевизначити метод onUpdateReceived(Update update) – який і реалізує основну логіку, він отримує обьект класу Update, який зберігає всю основну інформацію і вже потім виконує над ним якісь дії. Бот запускаться з функції main(), яка знаходиться в ньому же:

```
ApiContextInitializer.init();
    TelegramBotsApi telegramBotsApi = new TelegramBotsApi();
    try {
       telegramBotsApi.registerBot(Bot.getBot());
    }
```

У функції ініціалізується Апі контекст, створюється обьект класу TelegramBotsApi і запускається бот. Метод getBot() вертає новий обьект класу.

```
private static LongPollingBot getBot() {
    return new Bot();
  }
```

В конструкторі бота створюється обьект бази даних, виконується підключення та ініціалізуються списки команд: adminGroupCommands, adminPrivateCommands, curatorChatLogic, curatorPrivateLogic в яких зберігаються команди для певних груп користувачів відповідно до назви Set.

```
Bot () {
    db = new DatabaseManager("jdbc:mariadb://localhost:3306/test");
    this.db.connect();
    initCommands();
  }
```

Після обробки даних і виконання команди користувачу має надійти відповідь, яка надсилається за допомогою методів sendMsg(String chatId, String s, Update update) в котрому рядок з повідомленням обертається в обьект класу

SendMessage, та обрізається на декілька повідомлень, якщо довше ніж 4096 сиволів – стандартне обмеження Телеграм на довжину повідомлення. Фінальне повідомлення відсилається за допомогою sendMessage(sendMessage) класу від якого наслідуємося.

Для реалізації таймерів у классі Bot було створено поле

```
private final Map<Integer, Timer> timers = new HashMap<>();
```

в якому зберігається id користувача та таймер, який запускався для нього при надсиланні запитання в методі sendQuestion:

```
if (timers.containsKey(userId)) {
   timers.remove(userId);
}
Timer timer = new Timer();
BotTimerTask botTimerTask = new BotTimerTask(userId, questionId,
curTestId, update, this);
timer.schedule(botTimerTask, question.getTimer() * 1000);
timers.put(userId, timer);
```

Де BotTimerTask це класс який наслідується від TimerTask і в run методі завершає поточне запитання та відправляє нове запитання або завершає тест, якщо поточне запитання було фінальним.
Якщо ж система отримує відповідь на запитання до того, как сплине наданий час, то таймер відміняється:

```
if (timers.containsKey(userId)) {
   timers.get(userId).cancel();
}
```

Для реалізації адаптивного алгоритму розрахунку часових проміжків було створено клас AdaptiveTimerCounter, який наслідується від Thread.

У классі бот, при запуску тесту куратором стартується лічильник: AdaptiveTimerCounter counter = new AdaptiveTimerCounter(db,test_id); counter.start(); // start counter

В якому реалізована наступна логіка:

```
public void run() {
   Set<Integer> questions = db.getQuestionsOfTests(test_id).keySet();
   for (int quest_id:questions) {
      int timer = countTimerForQuestion(quest_id);
      db.getQuestionDB().updateTimer(quest_id, timer);
   }
}
```

Основна логіка підрахунку часу реалізована в методі:

```
public int countTimerForQuestion(int questionId) {
   Question question = db.getQuestionDB().getQuestion(questionId);
   int non_responded = 0;
   ArrayList<Integer> timediffs = new ArrayList<>();
   Set<Answer> answers =
db.doneTestAnswersDB.getAnswersByQuestionID(questionId);
   for (Answer answer: answers) {
      if (DbConstants.EMPTY_ANSWER.equals(answer.getAnswer())) {
         if (answer.getTimer() > question.getTimer()) { /* якщо людина не відповіла
за час, більший ніж поточний таймер, то підраховуємо його*/
            non_responded++;
         }
      } else {
         if (answer.is_right()) {
            timediffs.add(question.getTimer() - answer.getTimer());
         }
      }
   }
   double timediff = countMedian(timediffs); // підраховуємо медіану
   int new_timer;
   if ((double)non_responded/answers.size() > 0.3) { // якщо невідповівших більше
30%, то додаємо час
      new_timer= (int)(question.getTimer() + timediff * ( (double)
question.getTimer() / timediff) * 0.5);
```

```
    } else {
        new_timer = (int) ((double) question.getTimer() - timediff * (1 - (double)
non_responded / answers.size()) * 0.8);
    }
if (new_timer == 0) new_timer = question.getTimer();
    return new_timer;}
```

Метод countMedian реалізований для підрахунку медіани, код можна додатково переглянути у додатку.

3.4. Приклад тестування програми та результату роботи з програми

На рисунку 3.1 можна переглянути процес реэстрації користувача

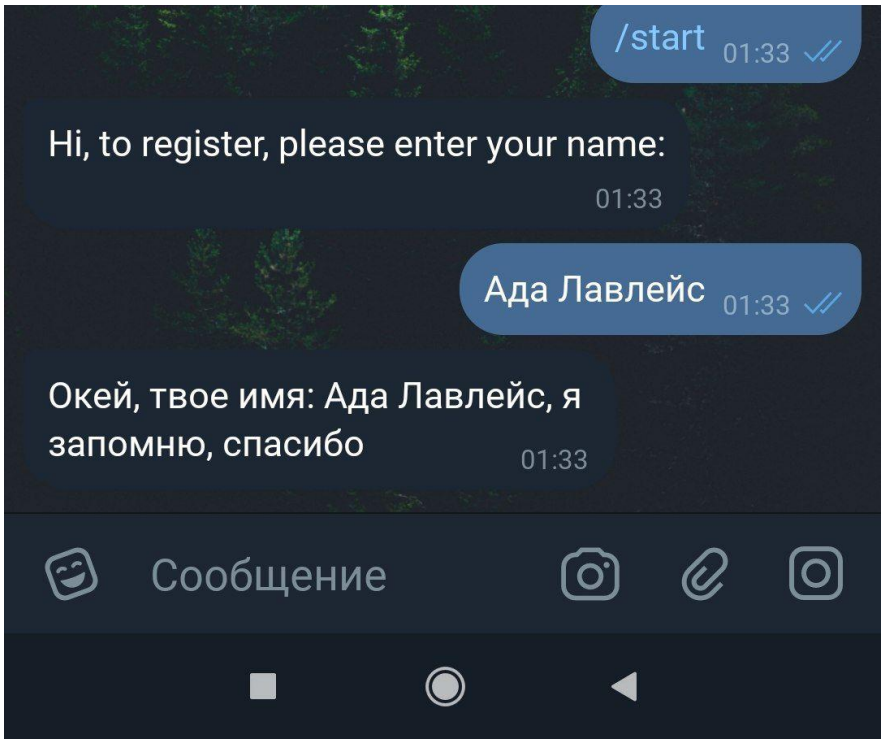


Рисунок 3.1 – реєстрація нового користувача

Адміністрування чату можна побачити на рис 3.2-3.3, а саме сворення чату, назначення юзерів кураторами, додавання незареєстрованих та зареєстрованих користувачів до групи.

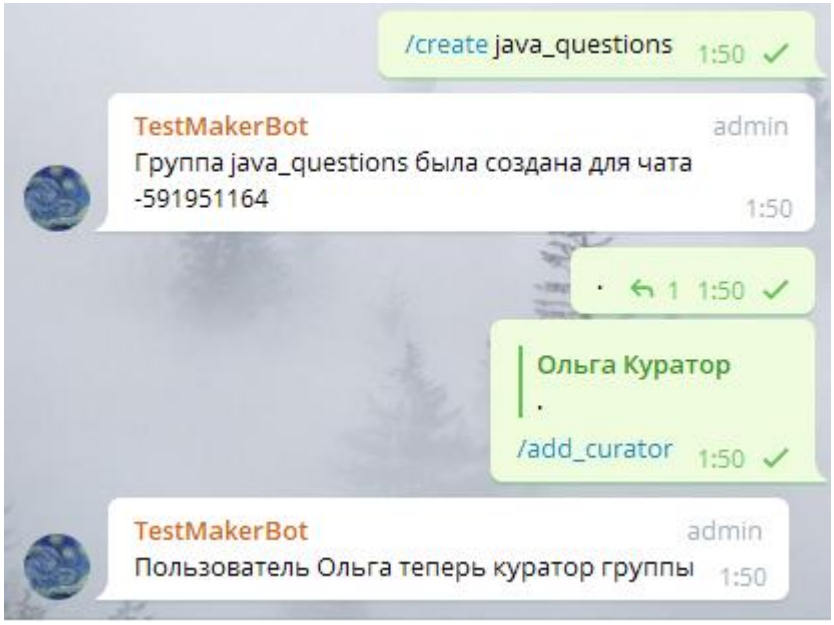


Рисунок 3.2 – адміністрування чату, створення куратора

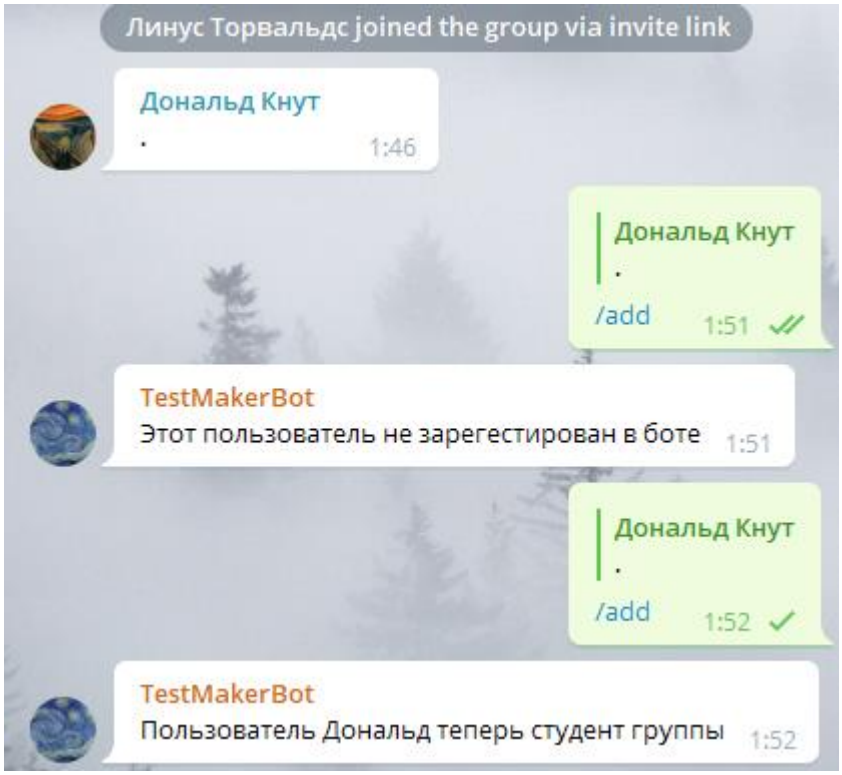


Рисунок 3.3 – додавання нових студентів до групи

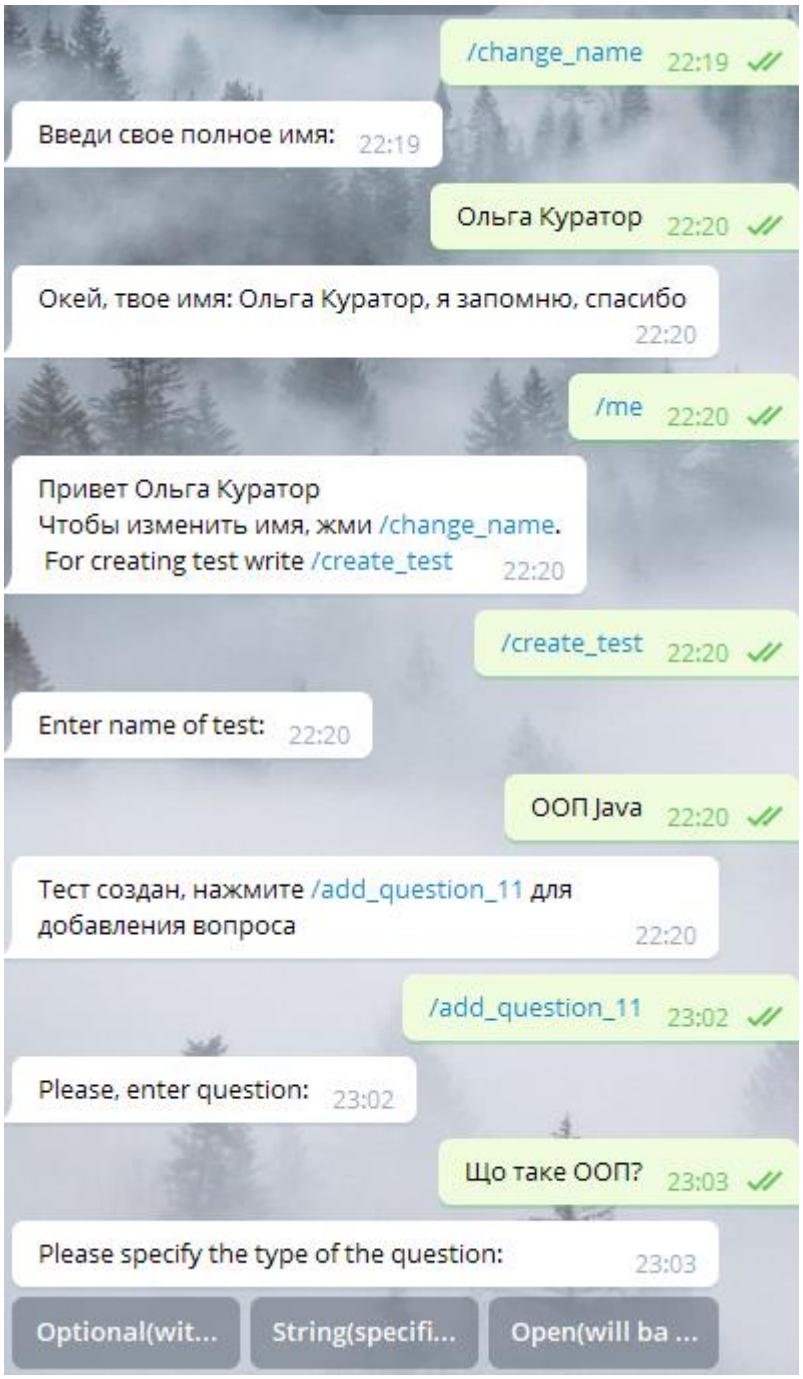


Рисунок 3.4 – алгоритм створення тесту у куратора

Куратор будь-якої групи може створити тест з декількаами видами питань, на рисунках 3.4-3.5 ми бачимо створення опціональних запитань (відповіді кнопками) та строкових (відповідь являється словом або строкою).

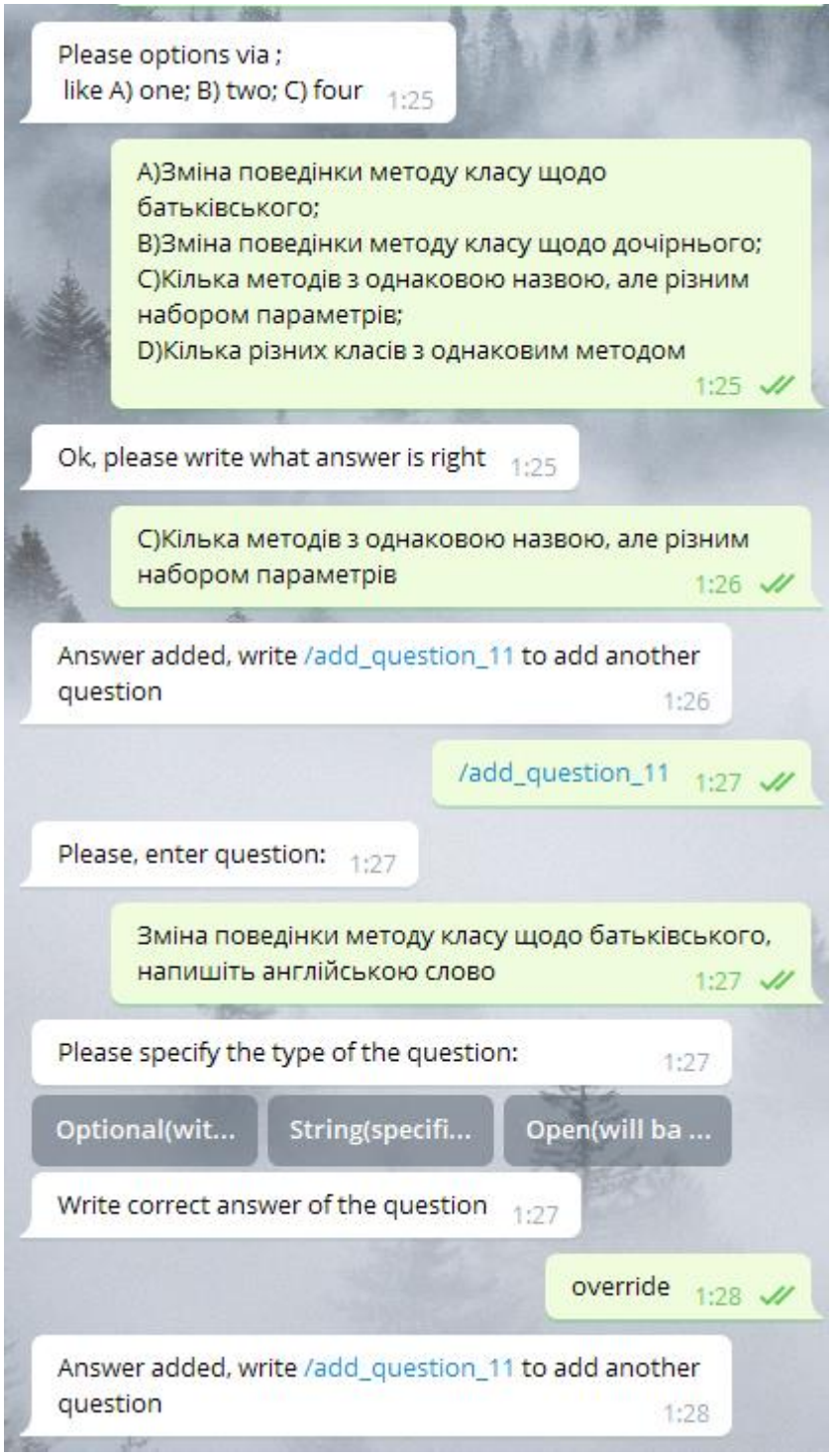


Рисунок 3.5 – додавання запитань(опціональні/строкові)

Тест починається куртором командою /start_test_ID, потрібну команду генерує бот, можна побачити на аутпуті команди /show_tests. Після закінчення тесту студентами, можна побачити результати командою /show_results_ID – рис 3.6-3.7.

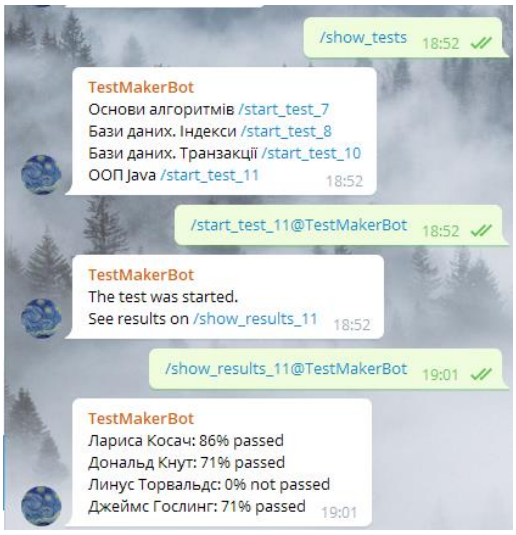


Рисунок 3.6 – початок тесту і перевірка результатів

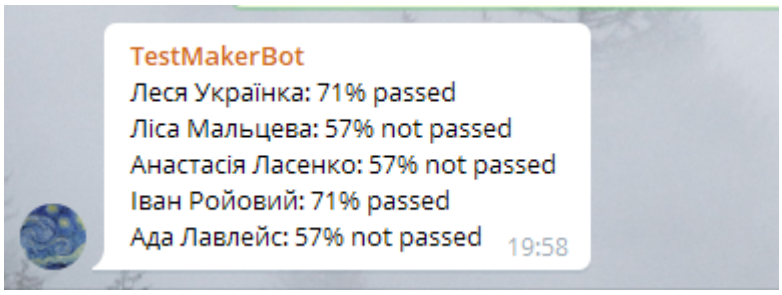


Рисунок 3.7- перегляд результатів для інших студентів

Якщо користувач не встиг відповісти на запитання (час, який відведений на запитання пишеться в тілі самого запитання, то бот надсилає відповідне повідомлення про те, що час сплинув (рис. 3.8) та надсилається наступне запитання. Запитання не зараховується студенту і враховується в результаті тесту, як провалене. Після завершення тесту, користувач отримує повідомлення з балами за тест та провалив він його чи ні (рис. 3.9).

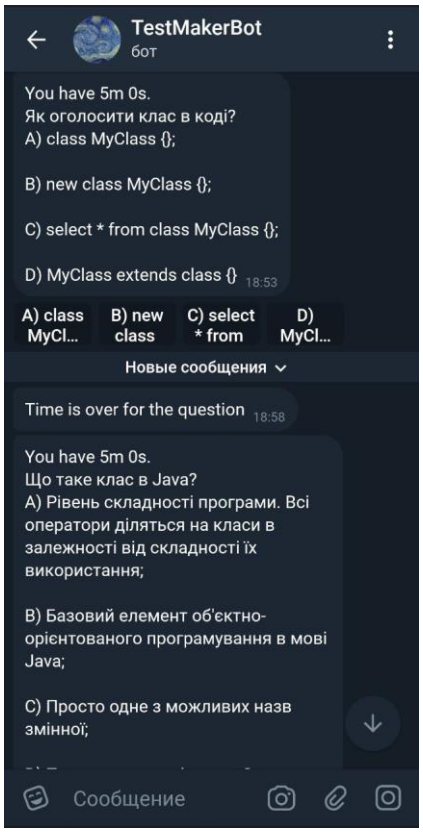


Рисунок 3.8 – час сплив для користувача

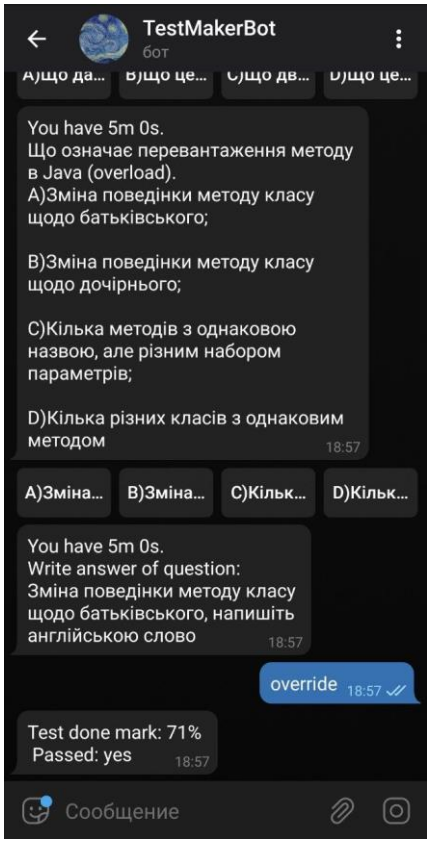
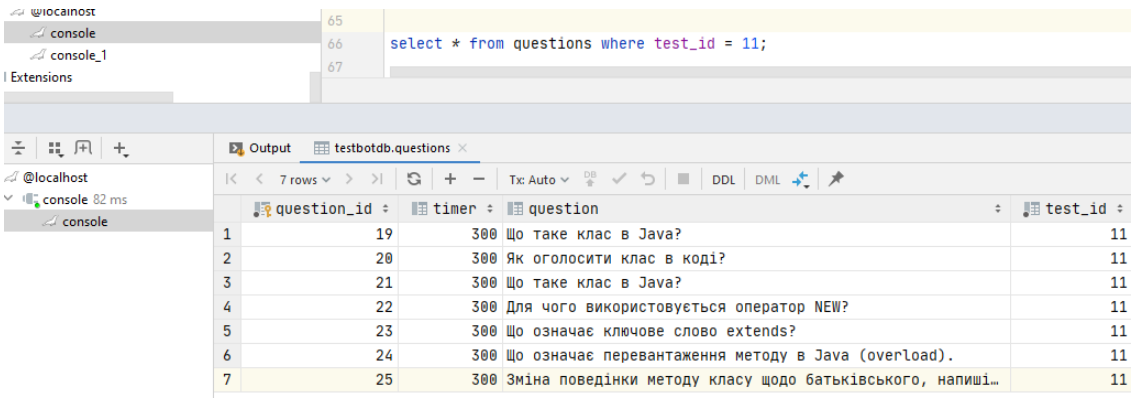


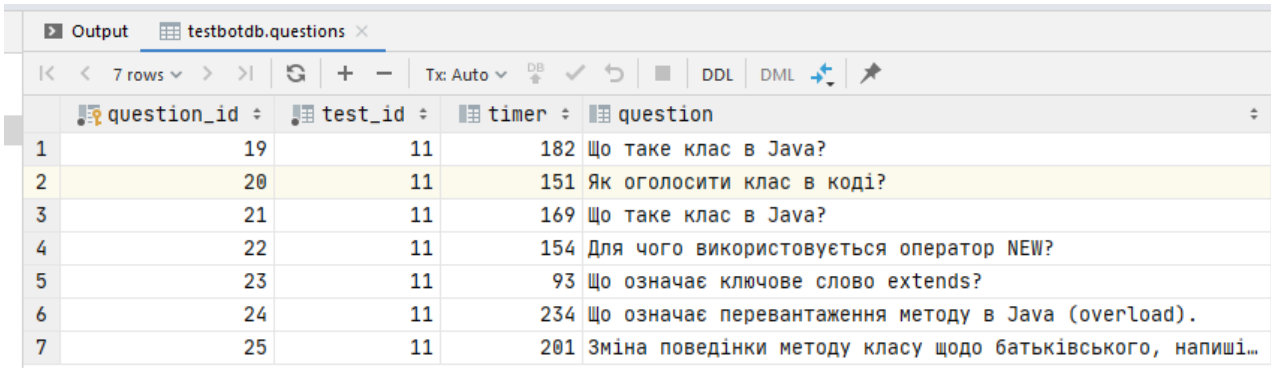
Рисунок 3.9 – завершення тесту для користувача

Дефолтно задається час на запитання 5 хвилин(рис. 3.10), при кожному старті тесту перераховуються таймери для кожного запитання на основі того, як відповідали попередні студенти. На рис. 3.11 можна побачити як змінилися таймери через проходження тесту декількаами групами студентів.



	question_id	timer	question	test_id
1	19	300	Що таке клас в Java?	11
2	20	300	Як оголосити клас в коді?	11
3	21	300	Що таке клас в Java?	11
4	22	300	Для чого використовується оператор NEW?	11
5	23	300	Що означає ключове слово extends?	11
6	24	300	Що означає перевантаження методу в Java (overload).	11
7	25	300	Зміна поведінки методу класу щодо батьківського, напиші…	11

Рисунок 3.10 – початковий список запитань для тесту



	question_id	test_id	timer	question
1	19	11	182	Що таке клас в Java?
2	20	11	151	Як оголосити клас в коді?
3	21	11	169	Що таке клас в Java?
4	22	11	154	Для чого використовується оператор NEW?
5	23	11	93	Що означає ключове слово extends?
6	24	11	234	Що означає перевантаження методу в Java (overload).
7	25	11	201	Зміна поведінки методу класу щодо батьківського, напиші…

Рисунок 3.11 – зміна таймерів після проходжень тесту декількома групами

Також для користувачів з ролью Адміна доступна додаткова команда, яка дозволяє надсилати запити напрямую до бази даних, приклад запиту можно побачити на рис. 3.12.

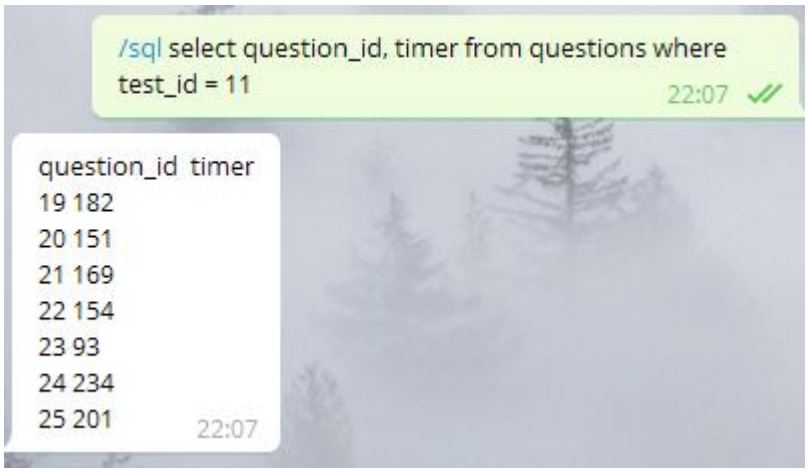


Рисунок 3.12 – адмін-виконання команд в боті

ВИСНОВКИ

В випускній роботі був реалізований алгоритм для підрахунку часових проміжків, що виділені студенту на виконання завдання. Було проаналізовано декілька способів та можливостей для вирішення проблеми та обрано власну реалізацію алгоритму як найбільш зручну та підходящу на даному сеті вхідних даних та кількості цих вхідних даних.

Для зручності підрахунку, запуск перераховування таймерів починався при старті тесту в окремому потоці параллельно з початком тесту у студентів, що мінімізує можливість пересічення запитів та блокування даних у таблицях.

Обмеження часу було реалізовано за допомогою таймерів, що запускалися для кожного запитання і сет с таймерами зберігається в темповій пам’яті додатку, тоді як запитання, на яке має відповідати користувач, зберігається уже в базі даних.

Автоматизована система тестування на базі месенджеру Телеграм була реалізована на мові Java, було використано роботу з JDBC модулем (запити до БД), роботу Телеграм Bot Api і реалізовано всі вимоги до системи. СУБД для ІС – MySQL, до неї була реалізована ЕР діаграма з урахованням особливостей СУБД (наприклад автоінкрементні поля) та вимог до збереження даних про таймери.

ЛІТЕРАТУРА

1. Опис мессенджера Телеграм [Електронний ресурс] / Режим доступу: https://tlgrm.ru/faq

2. Документація Телеграм по ботам [Електронний ресурс] / Режим доступу: https://tlgrm.ru/docs/bots

3. Документація Телеграм по Bot API [Електронний ресурс] / Режим доступу: https://tlgrm.ru/docs/bots/api

4. Вікіпедія, вільна енциклопедія, СУБД [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Database#Database_management_system

5. Вікіпедія, моделі даних [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Database_model

6. Інформаційний ресурс Habr [Електронний ресурс] / Режим доступу: https://habr.com/ru/company/ruvds/blog/324936/

7. Інформаційний ресурс Володимира Драча [Електронний ресурс] / Режим доступу: http://drach.pro/blog/hi-tech/item/145-db-comparison

8. Вікіпедія, Oracle Database [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Oracle_Database

9. Інформаційний ресурс Llost [Електронний ресурс] / Режим доступу: https://losst.ru/sravnenie-mysql-vs-mariadb

10. Knowledge Powerhouse, “Top 50 MySQL Interview Questions & Answers Kindle Edition”. Publisher “ Amazon Digital Services LLC”, 2016 – 44p.

11. Інформаційний ресурс Tproger [Електронний ресурс] / Режим доступу: https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/

12. Hutten D., “MySQL: MySQL Tutorials for Beginners Basic to Advanced MySQL Languages”. Publisher “Amazon Digital Services LLC”, 2018 – 122p.

13. MG Martin, "Java: Basic Fundamental Guide for Beginners Kindle Edition". Publisher “Amazon Digital Services LLC”, 2018 – 79c.

14. И.Н. Блинов, В.С. Романчик "Java. Методы программирования". Минск: издательство «Четыре четверти», 2013 – 896с.

15. Schildt H., “Java: A Beginner's Guide, Eighth Edition”. Publisher “McGraw-Hill Education”, 2018 – 1152p.

16. Schildt H., “Java: The Complete Reference, Eleventh Edition”. Publisher “McGraw-Hill Education”, 2018 – 1248p.

17. Давыдов С.В., Ефимов А.А. “Intellij Idea. Профессиональное программирование на Java. Санкт-Петербург: БХВ-Петербург, 2005. – 800с.

18. Tariq Rashid, “Make Your Own Neural Network”. Publisher “Amazon Digital Services LLC”, 2016 – 222p

19. Довбиш А.С., “Основи проектування інтелектуальних систем”, Сумський державний університет, 2009 – 178с

20. Автоматизована система тестування знань студентів на базі месенджера Телеграм, Бакалаврска робота, Литюга, О.Є. [Електронний ресурс] / Режим доступу: https://essuir.sumdu.edu.ua/handle/123456789/74637

ДОДАТОК А

Лістинг коду програми

```
package bot;

import model.database.DatabaseManager;
import model.database.DbConstants;
import model.entities.Answer;
import model.entities.Question;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Set;

public class AdaptiveTimerCounter extends Thread  {
   DatabaseManager db;
   int test_id;

   public AdaptiveTimerCounter(DatabaseManager databaseManager, int test_id) {
      this.db = databaseManager;
      this.test_id = test_id;
   }



   public int countTimerForQuestion(int questionId) {
      Question question = db.getQuestionDB().getQuestion(questionId);
      int non_responded = 0;
      ArrayList<Integer> timediffs = new ArrayList<>();
      Set<Answer> answers =  db.doneTestAnswersDB.getAnswersByQuestionID(questionId);
      for (Answer answer: answers) {
         if (DbConstants.EMPTY_ANSWER.equals(answer.getAnswer())) {
            if (answer.getTimer() > question.getTimer()) { //если неответивший человек не ответил за , большее
время, чем указаный таймер,
                                       // то его считаем
               non_responded++;
            }
         } else {
            if (answer.is_right()) {
               timediffs.add(question.getTimer() - answer.getTimer());
            }
         }
      }
      double timediff = countMedian(timediffs); //считаем медиану по ответившим на вопрос
      int new_timer;
      if ((double)non_responded/answers.size() > 0.3) { // если тех, кто не ответил больше 30%, то добавляем
время
         new_timer= (int)(question.getTimer() + timediff * ( (double) question.getTimer() / timediff) * 0.5);
      } else {
         new_timer = (int) ((double) question.getTimer() - timediff * (1 - (double) non_responded / answers.size()) *
0.8);
      }
      if (new_timer == 0) new_timer = question.getTimer();
      return new_timer;
   }
```

```
private double countMedian (List<Integer> list) {
   int median = 0;
   int[] data = new int[list.size()];
   for (int i = 0; i < list.size();  i++) {
      data[i] = list.get(i);
   }
   median = find(data);
   return median;
}


public static void swap(int[] a, int i1, int i2)
{
   int temp = a[i1];
   a[i1] = a[i2];
   a[i2] = temp;
}

public static int find(int[] a)
{
   if (a.length < 1) return 0;
   if (a.length < 2) return  a[0];
   int low = 0;
   int high = 0;
   int median = 0;
   int[] help = Arrays.copyOf(a, a.length);
   Arrays.sort(help);
   for (int i = 0; i < a.length; i++) {
      if (a[i] == help[0]) {
         low = i;
      }
   }
   for (int i = 0; i < a.length; i++) {
      if (a[i] == help[help.length - 1]) {
         high = i;
      }
   }
   for (int i = 0; i < a.length; i++) {
      if (a[i] == ((help[low] + help[high]) / 2)) {
         median = i;
      }
   }

   do
   {
      if (high <= low)
      {
         return a[median];
      }
      if (high == low + 1)
      {
         if (a[low] > a[high])
         {
            swap(a, low, high);
         }
         return a[median];
```

```
        }
        int middle = (low + high) / 2;
        if (a[middle] > a[high])
        {
           swap(a, middle, high);
        }
        if (a[low] > a[high])
        {
           swap(a, low, high);
        }
        if (a[middle] > a[low])
        {
           swap(a, middle, low);
        }
        swap(a, middle, low + 1);
        int ll = low + 1;
        int hh = high;
        do
        {
           do
           {
              ll++;
           }
           while(a[low] > a[ll]);
           do
           {
              hh--;
           }
           while(a[hh] > a[low]);
           if (hh < ll)
           {
              break;
           }
           swap(a, ll, hh);
        }
        while(true);
        swap(a, low, hh);
        if (hh <= median)
        {
           low = ll;
        }
        if (hh >= median)
        {
           high = hh - 1;
        }
     }
     while(true);
  }

  /**
   * When an object implementing interface <code>Runnable</code> is used
   * to create a thread, starting the thread causes the object's
   * <code>run</code> method to be called in that separately executing
   * thread.
   * <p>
   * The general contract of the method <code>run</code> is that it may
   * take any action whatsoever.
```

```
     *
     * @see Thread#run()
     */
    @Override
    public void run() {
        Set<Integer> questions = db.getQuestionsOfTests(test_id).keySet();
        for (int quest_id:questions) {
            int timer = countTimerForQuestion(quest_id);
            // System.out.println(quest_id + " - " + timer);
            db.getQuestionDB().updateTimer(quest_id, timer);
        }
    }
}

package bot;

import model.entities.Question;
import model.keyboards.*;
import model.database.*;
import org.mariadb.jdbc.internal.com.read.resultset.UpdatableColumnDefinition;
import org.telegram.telegrambots.ApiContextInitializer;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.send.SendSticker;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.User;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.meta.exceptions.TelegramApiRequestException;
import org.telegram.telegrambots.meta.generics.LongPollingBot;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.Timer;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


// если пользователь в тесте, запретить создавать опрос и наоборот

public class Bot extends TelegramLongPollingBot {

    final DatabaseManager db;
    private final KeyBoards keyBoards = new KeyBoards();
    private final Set<String> adminGroupCommands = new HashSet<>();
    private final Set<String> adminPrivateCommands = new HashSet<>();
    private final Set<String> curatorChatLogic = new HashSet<>();
    private final Set<String> curatorPrivateLogic = new HashSet<>();
    private final Set<String> userPrivateLogic = new HashSet<>();

    private final Map<Integer, Timer> timers = new HashMap<>();
```

```
    private final int MAX_TELEGRAM_MES_LENGTH = 4096;

    private final String BOT_TOKEN = "837238452:AAGIwAOsW9o4xOPR6SF7u5F699T4TIRJJbo";
   // BotLogger log = BotLogger.ge

    private static DocumentMarshaller actionMarshaller =  new JsonDocumentMarshallerImpl(Action.class, "Action");

    Bot () {
        db = new DatabaseManager("jdbc:mariadb://localhost:3306/test");
        this.db.connect();
        initCommands();
    }
    /**
     * Метод для приема сообщений.
     * @param update Содержит сообщение от пользователя.
     */
    @Override
    public void onUpdateReceived(Update update) {
        if (update.hasMessage()) {
            String command = "";
            if (update.getMessage().hasText()) {
                String text = update.getMessage().getText();
                 command = text.split(" ")[0];
                if (command.contains("@")) {
                    command = text.split("@")[0];
                    if (!getBotUsername().equals(text.split("@")[1])) command = "";
                }
                if (command.startsWith("/")) {
                    Pattern pattern = Pattern.compile("^(/\\w+_)\\d+$");
                    Matcher matcher = pattern.matcher(command);
                    if (matcher.find()) {
                        command = matcher.group(1);
                    }
                }

                if (update.getMessage().isGroupMessage() || update.getMessage().isSuperGroupMessage()) {
                    try {
                        groupChatLogic(update, command);
                    } catch (TelegramApiException e) {
                        e.printStackTrace();
                    }
                } else {
                    try {
                        privateChatLogic(update, command);
                    } catch (TelegramApiException e) {
                        e.printStackTrace();
                    }
                }
            }
        } else if (update.hasCallbackQuery()) {
            if (update.getCallbackQuery().getMessage().isGroupMessage() ||
update.getCallbackQuery().getMessage().isSuperGroupMessage()) {
                System.out.println("group");
            } else {
             //   System.out.println("privateCallback");
                try {
                    privateCallbackLogic(update);
```

```
                } catch (TelegramApiException e) {
                    e.printStackTrace();
                }
            }
        }
    // System.out.println(update);
    }

    public /*synchronized*/ void sendSticker(String chatId, String fileId) {
        SendSticker sendSticker = new SendSticker();
        sendSticker.setChatId(chatId);
        sendSticker.setSticker(fileId);
        try {
            execute(sendSticker);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }

    private /*synchronized*/ void groupChatLogic(Update update, String command) throws TelegramApiException {
        if (curatorChatLogic.contains(command) && db.isCuratorOfGroup(update.getMessage().getFrom().getId(),
update.getMessage().getChatId().intValue())) {
            curatorChatLogic(update, command);
        } else if (adminGroupCommands.contains(command) && db.checkRole(update.getMessage().getFrom().getId(),
Roles.ROLE_ADMIN)) {
            groupChatAdminLogic(update, command);
        }

    }

    private /*synchronized*/ void curatorChatLogic(Update update, String command) throws TelegramApiException {
        String group = db.groupIdExists(update.getMessage().getChatId().intValue());
        switch (command) {
            case Commands.ADD:
                StringBuilder mes;
                if (update.getMessage().isReply()) {
                    int replyUserId = update.getMessage().getReplyToMessage().getFrom().getId();
                    if (!db.existUser(replyUserId)) {
                        mes = new StringBuilder("Этот пользователь не зарегестирован в боте");
                    } else if (db.groupIdExists(update.getMessage().getChatId().intValue()) == null) {
                        mes = new StringBuilder("Группа для этого чата еще не была создана");
                    } else {
                        if (db.isStudentOfGroup(replyUserId, update.getMessage().getChatId().intValue())) {
                            mes = new StringBuilder("Пользователь " +
update.getMessage().getReplyToMessage().getFrom().getFirstName() + " уже студент группы");
                        } else {
                            db.setUserGroupRole(replyUserId, Roles.GROUP_ROLE_STUDENT,
update.getMessage().getChatId().intValue());
                            mes = new StringBuilder("Пользователь " +
update.getMessage().getReplyToMessage().getFrom().getFirstName() + " теперь студент группы");
                        }
                    }
                } else {
                    mes = new StringBuilder("It is not a reply!");
                }
                sendMsg(update.getMessage().getChatId().toString(), mes.toString());
                break;
```

```
        case Commands.CHAT:
            if (group != null) {
                sendMsg(update.getMessage().getChatId().toString(), "chat_id: " +
update.getMessage().getChatId().intValue() + "\n группа: " + group);
            }
            break;
        case Commands.SHOW_TESTS:
            HashMap<Integer, String> tests = db.getTestsForUser(update.getMessage().getFrom().getId());
            if (tests.isEmpty()) {
                sendMsg(update.getMessage().getChatId().toString(), "no tests");
            } else {
                mes = new StringBuilder();
                for (Map.Entry<Integer, String> entry:tests.entrySet()) {
                    mes.append(entry.getValue()).append("
").append(Commands.START_TEST).append(entry.getKey()).append("\n");
                }
                sendMsg(update.getMessage().getChatId().toString(), mes.toString());
            }
            break;
        case Commands.START_TEST:
            int test_id = parseDigitCommand(update.getMessage().getText());
            if (test_id >= 0) {
                ArrayList <Integer> students = db.getStudentsOfGroup(update.getMessage().getChatId().intValue());
                if (students.isEmpty()) {
                    mes = new StringBuilder("The group is emprty.");
                } else {
                    if (db.existTestForUser(update.getMessage().getFrom().getId(), test_id)) {
                        AdaptiveTimerCounter counter = new AdaptiveTimerCounter(db,test_id);
                        counter.start(); // start counter
                        startTestForStudents(students, test_id, update);
                        mes = new StringBuilder("The test was started. \nSee results on " + Commands.SHOW_RESULTS +
test_id);
                    } else {
                        mes = new StringBuilder("this test does not exist for you.");
                    }
                }
            } else  {
                mes = new StringBuilder("wrong command");
            }
            sendMsg(update.getMessage().getChatId().toString(), mes.toString());
            break;
        case Commands.SHOW_RESULTS:
            test_id = parseDigitCommand(update.getMessage().getText());
            if (test_id >= 0) {
                mes = new StringBuilder(db.getResultsOfTest(update.getMessage().getChatId().intValue(), test_id));
                if (mes.length() == 0) {
                    mes = new StringBuilder("no tests");
                }
            } else  {
                mes = new StringBuilder("wrong command");
            }
            sendMsg(update.getMessage().getChatId().toString(), mes.toString());
            break;
    }
  }

  private int parseDigitCommand(String text) {
```

```
        Pattern pattern = Pattern.compile("^/\\w+_(\\d+)");
        Matcher matcher = pattern.matcher(text);
        int res = -1;
        if (matcher.find()) {
            res = Integer.parseInt(matcher.group(1));
        }
        return res;
    }

    private void startTestForStudents (ArrayList <Integer> students , int test_id, Update update) throws
TelegramApiException {
        for (int userId:students ) {
            int cur_test = db.createCurrentTest(userId, test_id);
            int questionId = db.getQuestionIdOfCurrentTest(cur_test);
            db.updateState(userId, States.IN_TEST, cur_test, questionId);
            sendQuestion(userId, update, questionId, cur_test);
        }
    }

    public void sendQuestion(int userId, Update update, int questionId, int curTestId) throws TelegramApiException {
        String mes;
        Question question = db.getQuestionDB().getQuestion(questionId);
        if (!db.doneTestAnswersDB.checkExistsAnswer(questionId, curTestId, userId)) {
            db.doneTestAnswersDB.createDoneAnswer(questionId, curTestId, userId);
        }
        if (timers.containsKey(userId)) {
            timers.get(userId).cancel();
            timers.remove(userId);
        }
        Timer timer = new Timer();
        BotTimerTask botTimerTask = new BotTimerTask(userId, questionId, curTestId, update, this);
        timer.schedule(botTimerTask, question.getTimer() * 1000);
        timers.put(userId, timer);

        if (question == null) {
            mes = "Please contact admin";
            sendMsg(String.valueOf(userId), mes);
            //logger
        } else {
            if (question.getType().equals(QuestionTypes.VARIANT)) {
                ArrayList <InlineKeyboardButton> buttons = new ArrayList<>();
                String [] options = question.getData().split(";");
                for (String s: options) {
                    if (s != null && !s.trim().equals("")) {
                        buttons.add(keyBoards.button(s,s));
                    }
                }
                SendMessage sendMessage = keyBoards.getButtonMessage("You have "  +
parseSeconds(question.getTimer())  + "\n" + question.getQuestion() + "\n" + question.getData().replace(";", ";\n"),
                        String.valueOf(userId), buttons.toArray(new InlineKeyboardButton[buttons.size()]));
                execute(sendMessage);
            } else {
                mes = "You have "  + parseSeconds(question.getTimer()) + "\nWrite answer of question:\n" +
question.getQuestion();
                sendMsg(String.valueOf(userId), mes);
            }
```

```
            db.updateState(userId, States.IN_TEST, curTestId, questionId);
        }

    }

    /**
     * user is admin and writing in private chat
     * @param update
     * @param command
     */
    private /*synchronized*/ void groupChatAdminLogic(Update update, String command) {
        String text = update.getMessage().getText();
        String chatIdStr =  update.getMessage().getChatId().toString();
        switch (command) {
            case Commands.CREATE:
                Pattern pattern = Pattern.compile("^(?:" + Commands.CREATE + "|" + Commands.CREATE + "@" +
getBotUsername() + ")\\s+([\\w\\d]{2,})");
                Matcher matcher = pattern.matcher(text);
                if (matcher.find()) {
                    String groupName = matcher.group(1);
                    int chatId = update.getMessage().getChatId().intValue();
                    String fail = db.groupIdExists(chatId);
                    if (fail != null) {
                        sendMsg(chatIdStr, "Для данного чата уже создана группа: " + fail);
                    } else if (db.groupNameExists(groupName)) {
                        sendMsg(chatIdStr, "Группа " + groupName + " уже существует");
                    } else {
                        if (db.createGroup(chatId, groupName)) {
                            sendMsg(chatIdStr, "Группа " + groupName + " была создана для чата " + chatId);
                        } else {
                            sendMsg(chatIdStr, "ошибка");
                        }
                    }
                }
            break;
            case Commands.CHAT:
                String group = db.groupIdExists(update.getMessage().getChatId().intValue());
                if (group == null) {
                    sendMsg(chatIdStr, "chat_id: " + update.getMessage().getChatId() + "\n группа не создана");
                } else {
                    StringBuilder mes = new StringBuilder("chat_id: " + update.getMessage().getChatId() + "\n группа: " +
group);
                    ArrayList<Integer> curator = db.getCuratorsOfGroup(update.getMessage().getChatId().intValue());
                    if (curator.isEmpty()) {
                        mes.append("\nКураторы не назначены.");
                    } else {
                        mes.append("\nКураторы: ");
                        for (Integer user_id: curator) {
                            mes.append(" ").append(db.getUserName(user_id)).append(",");
                        }
                       mes.deleteCharAt(mes.length() - 1);
                    }
                    sendMsg(chatIdStr, mes.toString());
                }
            break;
            case Commands.SQL:
                String res = sqlQuery(text);
```

```
            sendMsg(update.getMessage().getChatId().toString(), res);
        break;
        case Commands.ADD_CURATOR:
            if (update.getMessage().isReply()) {
                int replyUserId = update.getMessage().getReplyToMessage().getFrom().getId();
                if (!db.existUser(replyUserId)) {
                    sendMsg(chatIdStr, "Этот пользователь не зарегестирован в боте");
                } else if (db.groupIdExists(update.getMessage().getChatId().intValue()) == null) {
                    sendMsg(chatIdStr, "Группа для этого чата еще не была создана");
                } else {
                    if (db.isCuratorOfGroup(replyUserId, update.getMessage().getChatId().intValue())) {
                        sendMsg(chatIdStr, "Пользователь " +
update.getMessage().getReplyToMessage().getFrom().getFirstName() + " уже был назначен куратором группы");
                    } else {
                        db.setUserGroupRole(replyUserId, Roles.GROUP_ROLE_CURATOR,
update.getMessage().getChatId().intValue());
                        sendMsg(chatIdStr, "Пользователь " +
update.getMessage().getReplyToMessage().getFrom().getFirstName() + " теперь куратор группы");
                    }
                }
            } else {
                sendMsg(chatIdStr, "It is not a reply!");
            }
        break;
        case Commands.DELETE_GROUP:
            db.deleteGroup(update.getMessage().getChatId().intValue());
            sendMsg(chatIdStr, "group was deleted");
        break;
    }
  }
  private /* synchronized */ void privateCallbackLogic(Update update) throws TelegramApiException {
    Message message = update.getCallbackQuery().getMessage();
    User user = update.getCallbackQuery().getFrom();
    int userId = user.getId();
    States userState = db.getUserState(userId);
    String mes = "";
    switch (userState) {
        case WAIT_TYPE:
            Action action = new ActionBuilder(actionMarshaller).build(update);
            QuestionTypes type = QuestionTypes.STRING;
            try {
                type = QuestionTypes.valueOf(action.getName());
            } catch (IllegalArgumentException e) {
                mes = "smth wrong, please contact admin";
                db.setState(userId, States.EMPTY);
            sendMsg(message.getChatId().toString(), mes);
            }
            db.setQuestType(db.getUsersQuestionId(userId), type.toString());
            if (type.equals(QuestionTypes.VARIANT)) {
                mes = "Please options via ;\n like A) one; B) two; C) four";
                db.setState(userId, States.WAIT_VARIANT);
            } else if(type.equals(QuestionTypes.STRING)) {
                mes = "Write correct answer of the question";
                db.setState(userId, States.WAIT_ANSWER);
            } else {
                int questId = db.getUsersQuestionId(userId);
                mes = "Answer added, write " + Commands.ADD_QUESTION  + db.getUsersTestId(questId) + " to add
```

```
another question";
            }
            sendMsg(message.getChatId().toString(), mes);
            break;
        case IN_TEST:

            Action action1 = new ActionBuilder(actionMarshaller).build(update);
            Question question = db.getQuestionDB().getQuestion(db.getUsersQuestionId(userId));
            if (QuestionTypes.VARIANT.equals(question.getType())) {
               String answer = "";
               String [] options = question.getData().split(";");
               for (String s: options) {
                  if (s != null && !s.trim().equals("")) {
                     if (s.contains(action1.getName())) {
                        answer = s;
                     }
                  }
               }
               db.doneTestAnswersDB.updateDoneAnswer(db.getUsersQuestionId(userId), db.getUsersTestId(userId),
answer, userId);
            }
            int curTestId = db.getUsersTestId(userId);
            int newQuest = getNewQuestId(userId);
            if (newQuest < 0) {
               endTest(userId, update);
            } else {
               if (timers.containsKey(userId)) {
                  timers.get(userId).cancel();
               }
               sendQuestion(userId, update, newQuest, curTestId);
            }
            break;
            default:
               break;
      }
   }

   public int getNewQuestId(int userId) {
      int curTestId = db.getUsersTestId(userId);
      int newQuest = db.getQuestionIdOfCurrentTest(curTestId);
     return newQuest;
   }

   private /* synchronized */  void privateChatLogic(Update update, String command) throws TelegramApiException {
      boolean isText = true;
      String mes = "";
      Message message = update.getMessage();
      User user = update.getMessage().getFrom();
      int userId = user.getId();
      States userState = db.getUserState(userId);
      switch (userState) {
         case WAIT_FOR_NAME:
            if (message.hasText()) {
               mes = message.getText();
               String nick = user.getUserName();
               System.out.println(nick + " " + mes);
               db.setUserName(userId, nick, mes);
```

```
                db.setState(userId, States.EMPTY);
                String name = db.getUserName(userId);
                mes = "Окей, твое имя: " + name + ", я запомню, спасибо";
            } else {
                mes = "Хм.. Что-то не так. Напиши мне текстом, пожалуйста";
            }
            break;
        case WAIT_FOR_TEST_NAME:
            if (db.hasCuratorRole(userId)) {
                if (message.hasText()) {
                    String text = update.getMessage().getText();
                    if (db.ifTestNameExists(text) < 0) {
                        int a = db.createTest(userId, text);
                        if (a > -1) {
                            mes = "Тест создан, нажмите " + Commands.ADD_QUESTION + a + " для добавления вопроса";
                            db.setState(userId, States.EMPTY);
                        } else {
                            mes = "smth wrong";
                        }
                    } else {
                        mes = "test with current name already exist";
                    }
                } else {
                    mes = "it is not a text";
                }
            } else {
                mes = "wrong permissions";
                db.setState(userId, States.EMPTY);
            }
            break;
        case WAIT_QUEST_NAME:
            if (message.hasText()) {
                db.addQuestion(db.getUsersQuestionId(userId), message.getText());
                db.setState(userId, States.WAIT_TYPE);
                        SendMessage sendMessage = keyBoards.getButtonMessage("Please specify the type of the
question:\n",
                        update.getMessage().getChatId().toString(),
                        keyBoards.button("Optional(with variants)", QuestionTypes.VARIANT.toString()),
                        keyBoards.button("String(specific string answer)", QuestionTypes.STRING.toString()),
                        keyBoards.button("Open(will ba checked\nby teacher", QuestionTypes.OPEN.toString()));
                execute(sendMessage);
            } else {
                mes = "smth wrong, enter text.";
            }
            break;
        case WAIT_ANSWER:
            if (message.hasText()) {
                int questId = db.getUsersQuestionId(userId);
                db.addAnswerToQuestion(questId, message.getText());
                mes = "Answer added, write " + Commands.ADD_QUESTION  + db.getUsersTestId(userId) + " to add
another question";
                db.setState(userId, States.EMPTY);
            } else {
                mes = "smth wrong, enter text.";
            }
            break;
        case WAIT_VARIANT:
```

```
            String text = update.getMessage().getText();
            if ( text.trim().contains(";")) {
                mes = "Ok, please write what answer is right";
                db.setQuestData(db.getUsersQuestionId(update.getMessage().getFrom().getId()), text);
                db.setState(userId, States.WAIT_ANSWER);
            } else {
                db.setState(userId, States.WAIT_VARIANT);
                mes = "Please options via ;\n like A) one; B) two; C) four";
            }
            break;
        case IN_TEST:
            if (message.hasText()) {
                int questId = db.getUsersQuestionId(userId);
                int curTestId = db.getUsersTestId(userId);
                Question question = db.getQuestionDB().getQuestion(questId);
                if (QuestionTypes.VARIANT.equals(question.getType())) {
                    mes = "Please, push the button";
                    sendQuestion(userId, update, questId, curTestId);
                } else {
                    String answer = message.getText();
                    db.doneTestAnswersDB.updateDoneAnswer(questId, curTestId, answer, userId);
                    int newQuest = db.getQuestionIdOfCurrentTest(curTestId);
                    if (newQuest < 0) {
                        endTest(userId, update);
                    } else {
                        if (timers.containsKey(userId)) {
                            timers.get(userId).cancel();
                        }
                        sendQuestion(userId, update, newQuest, curTestId);
                    }
                    return;
                }
            } else {
                mes = "Enter text, please.";
            }
            break;
        case EMPTY:
            boolean exist = db.existUser(userId);
            if (!exist) {
             //  sendSticker(message.getChatId().toString(), "CAADAgADZR0AAulVBRjcV5e5KVl7-wI");
                mes = "Hi, to register, please enter your name: ";
                db.setState(userId, States.WAIT_FOR_NAME);
                db.setUserName(userId, user.getUserName(), user.getFirstName()  + (user.getLastName()!= null? " "
+user.getLastName():""));
            } else {
                if (adminPrivateCommands.contains(command)) {
                    if (db.checkRole(userId, Roles.ROLE_ADMIN)) {
                        if (Commands.SQL.equals(command)) {
                            String res = sqlQuery(message.getText());
                            sendMsg(update.getMessage().getChatId().toString(), res);
                        } else {
                            sendMsg(update.getMessage().getChatId().toString(), "unknown command");
                        }
                    }
                } else if (curatorPrivateLogic.contains(command)) {
                    if (db.hasCuratorRole(userId)) {
                        switch (command) {
```

```
                    case Commands.CREATE_TEST:
                        mes = "Enter name of test: ";
                        db.setState(userId, States.WAIT_FOR_TEST_NAME);
                        break;
                    case Commands.ADD_QUESTION:
                        Pattern pattern = Pattern.compile("^/\\w+_(\\d+)$");
                        Matcher matcher = pattern.matcher(message.getText());
                        int test_id;
                        if (matcher.find()) {
                            test_id = Integer.parseInt(matcher.group(1));
                            if (db.existTestForUser(userId, test_id)) {
                                int question_id = db.createQuestion(test_id);
                                if (question_id > 0) {
                                    db.updateState(userId, States.WAIT_QUEST_NAME, test_id, question_id);
                                    mes = "Please, enter question: ";
                                } else {
                                    mes = "mistake";
                                }
                            } else {
                                mes = "You dont have permissions ro change this test";
                            }
                        } else {
                            mes = "wrong command";
                        }
                        break;
                }
            } else {
                mes = "You don't have permissions.";
            }
        } else if (userPrivateLogic.contains(command)){
            switch (command) {
                case Commands.CHANGE_NAME:
                    db.setState(userId, States.WAIT_FOR_NAME);
                    mes  = "Введи свое полное имя:";
                    break;
            }
        } else {
            mes = "Привет " + db.getUserName(userId) +  "\nЧтобы изменить имя, жми /change_name.";
            if (db.hasCuratorRole(userId)) {
                mes += "\n For creating test write " + Commands.CREATE_TEST;
            }
        }
    }
    break;
  }
  if (isText)
  sendMsg(message.getChatId().toString(), mes);
}

void endTest(int userId, Update update) {
  int cur_test = db.getUsersTestId(userId);
  int testId = db.getTestIdOfDoneTests(cur_test);
  HashMap<Integer,String> questions = db.getQuestionsOfTests(testId);
  HashMap<Integer,String> answers = db.getAnswersOfDoneTests(cur_test);
  int sum = 0;
  for (Map.Entry<Integer, String> question:questions.entrySet()) {
    String answer = answers.get(question.getKey()).trim().toLowerCase();
```

```
            if (question.getValue().trim().toLowerCase().equals(answer)) {
                sum += 1;
            }
        }
        if (timers.containsKey(userId)) {
            timers.get(userId).cancel();
            timers.remove(userId);
        }
        int res = Math.round(((float) sum / questions.size()) * 100);
        int passed = res > 60 ? 1 : -1;
        db.getUserStateDB().doneTest(cur_test, res, passed);
        db.setState(userId, States.EMPTY);
        String mes = "Test done mark: " + res + "%\n Passed: " + (passed > 0 ? "yes" : "no");
        sendMsg(String.valueOf(userId), mes);
    }

    public /*synchronized*/ String sqlQuery(String sql) {
        String result;
        sql = sql.substring(sql.indexOf(" "));
        result = db.sqlQuery(sql);
        return result;
    }

    /**
     * Метод для настройки сообщения и его отправки.
     * @param chatId id чата
     * @param s Строка, которую необходимот отправить в качестве сообщения.
     */
    public /* synchronized */ void sendMsg(String chatId, String s) {
        if (s==null || s.trim().equals("")) return;
        SendMessage sendMessage = new SendMessage();
        sendMessage.enableMarkdown(true);
        sendMessage.setChatId(chatId);
        System.out.println(chatId + ": " + s);
        sendMessage.enableMarkdown(false);
        while (s.length() >= MAX_TELEGRAM_MES_LENGTH) {
            sendMessage.setText(s.substring(0, MAX_TELEGRAM_MES_LENGTH));
            s = s.substring(MAX_TELEGRAM_MES_LENGTH);
        }
        sendMessage.setText(s);
        try {
            execute(sendMessage);
        } catch (TelegramApiException e) {
            System.out.println("Exception: " +  e.toString());
        }
    }

    private String parseSeconds(int sec) {
        String seconds = null;
        int min = (sec / 60)%60;
        int hours = (sec/60)/60;
        seconds = (hours>0?(hours + "h "):"") + (min>0?(min + "m "):"") + (sec-min*60-hours*60*60) + "s.";
        return  seconds;
    }


    /**
```

```
    * Метод возвращает имя бота, указанное при регистрации.
    * @return имя бота
    */
   @Override
   public String getBotUsername() {
      return "TestMakerBot";
   }



   public static void main(String[] args) {
      ApiContextInitializer.init();
      TelegramBotsApi telegramBotsApi = new TelegramBotsApi();
      try {
         telegramBotsApi.registerBot(Bot.getBot());
      } catch (TelegramApiRequestException e) {
         e.printStackTrace();
      }
   }

   private static LongPollingBot getBot() {
      return new Bot();
   }

   private void initCommands() {
      adminGroupCommands.add(Commands.CREATE);
      adminGroupCommands.add(Commands.ADD);
      adminGroupCommands.add(Commands.CHAT);
      adminGroupCommands.add(Commands.SQL);
      adminGroupCommands.add(Commands.ADD_CURATOR);

      adminPrivateCommands.add(Commands.DELETE_GROUP);
      adminPrivateCommands.add(Commands.SQL);

      curatorChatLogic.add(Commands.ADD);
      curatorChatLogic.add(Commands.START_TEST);
      curatorChatLogic.add(Commands.CHAT);
      curatorChatLogic.add(Commands.SHOW_TESTS);
      curatorChatLogic.add(Commands.SHOW_RESULTS);

      curatorPrivateLogic.add(Commands.CREATE_TEST);
      curatorPrivateLogic.add(Commands.ADD_QUESTION);

      userPrivateLogic.add(Commands.CHANGE_NAME);
   }

   /**
    * Метод возвращает token бота для связи с сервером Telegram
    * @return token для бота
    */
   @Override
   public String getBotToken() {
      return BOT_TOKEN;
   }
}
```

```
package bot;

import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.util.TimerTask;

import static model.database.DbConstants.EMPTY_ANSWER;

public class BotTimerTask extends TimerTask {
    int userId;
    int questionId;
    int curTestId;
    Update update;

    Bot bot;

    public BotTimerTask(int userId, int questionId, int curTestId, Update update, Bot bot) {
        super();
        this.userId = userId;
        this.questionId = questionId;
        this.curTestId = curTestId;
        this.update = update;
        this.bot = bot;
    }

    /**
     * The action to be performed by this timer task.
     */
    @Override
    public void run() {
        bot.db.doneTestAnswersDB.updateDoneAnswer(questionId, curTestId, EMPTY_ANSWER, userId);
        //int userId, Update update, int questionId, int curTestId
        try {
            bot.sendMsg(String.valueOf(userId), "Time is over for the question");
            int newQuest = bot.getNewQuestId(userId);
            if (newQuest < 0) {
                bot.endTest(userId, update);
            } else {
                bot.sendQuestion(userId, update, newQuest, curTestId);
            }
        } catch (TelegramApiException e) {
            e.printStackTrace();
        } finally {
            this.cancel();
        }
    }
}

package bot;

public class Commands {
    public final static String ADD_QUESTION = "/add_question_";
    public final static String START_TEST = "/start_test_";
    public final static String CREATE = "/create";
    public final static String CHAT = "/chat";
    public final static String ADD_CURATOR = "/add_curator";
```

```
    public final static String ADD = "/add";
    public final static String GRANT_MODER = "/grant_moder";
    public final static String SQL = "/sql";
    public final static String DELETE_GROUP = "/delete_group";
    public final static String CREATE_TEST = "/create_test";
    public final static String CHANGE_NAME = "/change_name";
    public final static String ADD_VARIANT = "/add_variant";
    public final static String DONE_QUESTION = "/done_question";
    public final static String SHOW_TESTS = "/show_tests";
    public final static String SHOW_RESULTS = "/show_results_";
}
package bot;

public enum QuestionTypes {
    VARIANT, OPEN, STRING
}

package bot;

public class Roles {
    public final static String ROLE_ADMIN = "ADMIN";
    public final static String ROLE_MODER = "MODER";

    public final static String GROUP_ROLE_CURATOR = "CURATOR";
    public final static String GROUP_ROLE_STUDENT = "STUDENT";
}

package bot;

public enum States {
    WAIT_FOR_NAME, IN_TEST, EMPTY, WAIT_FOR_TEST_NAME, WAIT_QUEST_NAME, WAIT_ANSWER, WAIT_TYPE,
WAIT_VARIANT
}
package model.database;

import bot.Roles;
import bot.States;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.ArrayList;
import java.util.HashMap;

import static model.database.DbConstants.*;

public class DatabaseManager {
    private Connection c;
    private final String dbUrl;

    static final String JDBC_DRIVER = "org.mariadb.jdbc.Driver";
```

```
static final String USER = "root";
static final String PASS = "root";

private QuestionTestDB questionTestDB;
private UserStateDB userStateDB;
public DoneTestAnswersDB doneTestAnswersDB;


public DatabaseManager(String dbUrl) {
   this.dbUrl = dbUrl;
   questionTestDB = new QuestionTestDB(this);
   userStateDB = new UserStateDB(this);
   doneTestAnswersDB = new DoneTestAnswersDB(this);
}

public void connect() {
   try {
      Class.forName(JDBC_DRIVER);
      c = DriverManager.getConnection(dbUrl, USER, PASS);
      System.out.println("Connecting to a selected model.database...");
      dbInit();
   } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
   }
}

private void dbInit() {
   try {
      checkConnection();
      Statement stmt = c.createStatement();
      stmt.executeUpdate(CREATE_DB);
      stmt.execute(USE_BD);
      stmt.execute("SET NAMES 'utf8mb4' COLLATE 'utf8mb4_unicode_ci'");
      stmt.executeUpdate(CREATE_USERS_TABLE_STATEMENT);
      stmt.executeUpdate(CREATE_STATES_TABLE);
      stmt.executeUpdate(CREATE_GROUPS_TABLE);
      stmt.executeUpdate(CREATE_USERS_GROUPS_TABLE);
      stmt.executeUpdate(CREATE_TESTS_TABLE);
      stmt.executeUpdate(CREATE_QUEST_TABLE_STATEMENT);
      stmt.executeUpdate(CREATE_VARIANTS_TABLE);
      stmt.executeUpdate(CREATE_DONE_TESTS_TABLE);
      stmt.executeUpdate(CREATE_ANSWERS_TABLE_STATEMENT);
      stmt.close();
   } catch (SQLException e) {
      e.printStackTrace();
   }

}

public Connection getConnection() {
   try {
      checkConnection();
   } catch (SQLException throwables) {
      throwables.printStackTrace();
   }
   return c;
```

```
    }

    private void checkConnection() throws SQLException {
        if (c.isClosed())
            connect();
    }

    public QuestionTestDB getQuestionDB() {
        return questionTestDB;
    }

    public UserStateDB getUserStateDB() {
        return userStateDB;
    }

    public boolean existUser(Integer userId) {
        boolean res = false;
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select 1 from " +
DbConstants.USERS_TABLE_NAME + " where " + COL_USER_ID + " = ?");
            preparedStatement.setInt(1, userId);
            ResultSet resultSet = preparedStatement.executeQuery();
            res = resultSet.next();
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }


    public States getUserState(Integer userId) {
        States state = States.EMPTY;
        String sql = "select " + COL_STATE + " from " + STATES_TABLE_NAME +
                " where " + COL_USER_ID + " = ?";
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, userId);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                state = States.valueOf(resultSet.getString(1));
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return state;
    }

    public String getUserName(Integer userId) {
        String name = "failure";
        String sql = "select " + COL_NAME + " from " + USERS_TABLE_NAME +
                " where " + COL_USER_ID + " = ?";
```

```
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, userId);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                name = resultSet.getString(1);
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return name;
    }

    public boolean setState(Integer userId, States state) {
        int ok;
        String sql = "insert into " + DbConstants.STATES_TABLE_NAME + " (" + COL_USER_ID + ", " + COL_STATE + ") " +
                " values ( ?, ?) ON DUPLICATE KEY UPDATE " + DbConstants.COL_STATE + " = ?";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, userId);
            prstmnt.setString(2, state.name());
            prstmnt.setString(3, state.name());
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }
    public boolean updateState(Integer userId, States state, int test_id, int question_id) {
        int ok;
        String sql = "insert into " + DbConstants.STATES_TABLE_NAME + " (" + COL_USER_ID + ", " + COL_STATE + ", " +
COL_TEST_ID + ", " + COL_QUESTION_ID + ") " +
                " values ( ?, ?, ?, ?)  ON DUPLICATE KEY UPDATE " + COL_STATE + " = ?, "  + COL_TEST_ID + " = ?, " +
COL_QUESTION_ID + " = ? ";


        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1,userId);
            prstmnt.setString(2, state.name());
            prstmnt.setInt(3,test_id );
            prstmnt.setInt(4,question_id );
            prstmnt.setString(5, state.name());
            prstmnt.setInt(6,test_id );
            prstmnt.setInt(7,question_id );
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
```

```
    return ok > 0;
}

public boolean setUserName(Integer userId, String nick, String name) {
    boolean ok;
    String sql = "insert into " + DbConstants.USERS_TABLE_NAME + "( " +
            COL_USER_ID + ", " + COL_NICK + ", " + COL_NAME + " )" +
            "values ( ?, ?, ?) ON DUPLICATE KEY UPDATE " + DbConstants.COL_NICK + " = ?, " + COL_NAME + " = ? ";
    try {
        checkConnection();
        PreparedStatement prstmnt = c.prepareStatement(sql);
        prstmnt.setInt(1, userId);
        if (nick != null) {
            prstmnt.setNString(2, nick);
            prstmnt.setNString(4, nick);
        } else {
            prstmnt.setNull(2, Types.CHAR);
            prstmnt.setNull(4, Types.CHAR);
        }
        if (name != null && name.length() > GREAT_CHAR) {
            name = name.substring(0, GREAT_CHAR - 1);
        }
        prstmnt.setNString(3, name == null ? "Unknown wanderer" : name);
        prstmnt.setNString(5, name == null ? "Unknown wanderer" : name);
        ok = prstmnt.execute();
        prstmnt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return ok;
}

public boolean groupNameExists(String groupName) {
    boolean ok = false;
    String sql = "select " + COL_NAME + " from " + GROUPS_TABLE_NAME + " where " + COL_NAME + " = ?";
    try {
        checkConnection();
        PreparedStatement prstmnt = c.prepareStatement(sql);
        prstmnt.setString(1, groupName);
        ResultSet resultSet = prstmnt.executeQuery();
        if (resultSet.next()) {
            ok = true;
        }
        resultSet.close();
        prstmnt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return ok;
}

public String groupIdExists(int id) {
    String res = null;
    String sql = "select " + COL_NAME + " from " + GROUPS_TABLE_NAME + " where " + COL_CHAT_ID + " = ?";
    try {
        checkConnection();
```

```
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, id);
            ResultSet resultSet = prstmnt.executeQuery();
            if (resultSet.next()) {
                res = resultSet.getString(1);
            }
            resultSet.close();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }

    public boolean createGroup(int id, String name) {
        int ok;
        String sql = "insert into " + DbConstants.GROUPS_TABLE_NAME + " (" + COL_NAME + ", " + COL_CHAT_ID + ") " +
                " values ( ?, ?) ON DUPLICATE KEY UPDATE " + COL_NAME + " = ?";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setString(1, name);
            prstmnt.setInt(2, id);
            prstmnt.setString(3, name);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            ok = -1;
        }
        return ok > 0;
    }

    public boolean checkRole(int userId, String role) {
        boolean ok = false;
        String sql = "select 1 from " + USERS_TABLE_NAME + " where " + COL_USER_ID + "= ? and " + COL_ROLE + "= ?";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, userId);
            prstmnt.setString(2, role);
            ResultSet resultSet = prstmnt.executeQuery();
            if (resultSet.next()) {
                ok = true;
            }
            resultSet.close();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ok;
    }

    public boolean setUserGroupRole(int userId, String role, int chat_id) {
        int ok;
        String sql = "insert into " + DbConstants.US_GROUP_TABLE_NAME + " (" + COL_USER_ID + ", " + COL_ROLE + ", " +
COL_CHAT_ID + ") " +
```

```
            " values ( ?, ?, ?)";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, userId);
            prstmnt.setString(2, role);
            prstmnt.setInt(3, chat_id);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            ok = -1;
        }
        return ok > 0;
    }

/*
    public boolean deleteUserGroupRole(int userId, String role, int chat_id) {
        int ok;
        String sql = "delete from " + US_GROUP_TABLE_NAME + " where "
                + COL_USER_ID + " = ? and  " + COL_ROLE + " = ? and  " + COL_CHAT_ID + " = ? ";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, userId);
            prstmnt.setString(2, role);
            prstmnt.setInt(3, chat_id);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            ok = -1;
        }
        return ok > 0;
    }
*/
 /*  private boolean grantRole(int userId, String role) {
        return true;
    }

    public boolean grantModer(int userId) {
        return grantRole(userId, Roles.ROLE_MODER);
    } */

    private boolean hasGroupRole(int userId, String role) {
        boolean res = false;
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select 1 from " + US_GROUP_TABLE_NAME + "
where " + COL_USER_ID + " = ? AND " + COL_ROLE + " = ?");
            preparedStatement.setInt(1, userId);
            preparedStatement.setString(2, role);
            ResultSet resultSet = preparedStatement.executeQuery();
            res = resultSet.next();
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return res;
    }

    public boolean hasCuratorRole(int userId) {
        return hasGroupRole(userId, Roles.GROUP_ROLE_CURATOR);
    }

    public String sqlQuery(String query) {
        String result = "";
        if (query == null) {
            result = "null";
        } else {
            try {
                checkConnection();
                PreparedStatement prstmnt = c.prepareStatement(query);
                ResultSet resultSet;
                if (query.toLowerCase().trim().startsWith("select")) {
                    resultSet = prstmnt.executeQuery();
                    StringBuilder stringBuffer = new StringBuilder();
                    if (resultSet.next()) {
                        ResultSetMetaData rsmd = resultSet.getMetaData();
                        for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                            stringBuffer.append(rsmd.getColumnName(i));
                            stringBuffer.append("  ");
                        }
                        do {
                            stringBuffer.append("\n");
                            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                                int type = rsmd.getColumnType(i);
                                if (type == Types.CHAR || type == Types.VARCHAR) {
                                    stringBuffer.append(resultSet.getString(i));
                                } else if (type == Types.DATE) {
                                    stringBuffer.append(resultSet.getDate(i).toString());
                                } else if (type == Types.BLOB) {
                                    stringBuffer.append(resultSet.getBlob(i).toString());
                                } else if (type == Types.INTEGER) {
                                    stringBuffer.append(resultSet.getInt(i));
                                } else {
                                    stringBuffer.append("UNKNOWN TYPE");
                                }
                                stringBuffer.append(" ");
                            }
                        } while (resultSet.next());
                        result = stringBuffer.toString();
                    } else {
                        result = "no rows selected";
                    }
                    resultSet.close();
                } else {
                    prstmnt.execute();
                    result = "executed";
                }
                prstmnt.close();
            } catch (SQLException e) {
                result += e.getStackTrace();
```

```
            }
        }
        return result;
    }



    public boolean isCuratorOfGroup(int userId, int chatId) {
      return isCurrentRoleOfGroup(userId,chatId, Roles.GROUP_ROLE_CURATOR);
    }

    public boolean isStudentOfGroup(int userId, int chatId) {
        return isCurrentRoleOfGroup(userId,chatId, Roles.GROUP_ROLE_STUDENT);
    }

    private boolean isCurrentRoleOfGroup(int userId, int chatId, String role) {
        boolean res = false;
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select 1 from " + US_GROUP_TABLE_NAME + "
where " + COL_USER_ID + " = ? AND " + COL_CHAT_ID + " = ? and " + COL_ROLE + " = ?");
            preparedStatement.setInt(1, userId);
            preparedStatement.setInt(2, chatId);
            preparedStatement.setString(3, role);
            ResultSet resultSet = preparedStatement.executeQuery();
            res = resultSet.next();
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }

    private ArrayList<Integer> getUserRoleOfGroup(int chatId, String role) {
        ArrayList<Integer> res = new ArrayList<>();
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select " + COL_USER_ID +
                    " from " + US_GROUP_TABLE_NAME + " where "
                    + COL_CHAT_ID + " = ? and " + COL_ROLE + " = ?" );
            preparedStatement.setInt(1, chatId);
            preparedStatement.setString(2, role);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                res.add(resultSet.getInt(1));
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }

    public ArrayList<Integer> getCuratorsOfGroup(int chatId) {
      return getUserRoleOfGroup(chatId, Roles.GROUP_ROLE_CURATOR);
```

```
}


public ArrayList<Integer> getStudentsOfGroup(int chatId) {
   return getUserRoleOfGroup(chatId, Roles.GROUP_ROLE_STUDENT);
}



public boolean deleteGroup(int chatId) {
   boolean ok = true;
   String sql = "delete from " + US_GROUP_TABLE_NAME + " where "
         +  COL_CHAT_ID + " = ? ";
   String sql1 = "delete from " + GROUPS_TABLE_NAME + " where "
         +  COL_CHAT_ID + " = ? ";
   try {
      checkConnection();
      PreparedStatement prstmnt = c.prepareStatement(sql);
      prstmnt.setInt(1, chatId);
      prstmnt.executeUpdate();
      prstmnt.close();
      prstmnt = c.prepareStatement(sql1);
      prstmnt.setInt(1, chatId);
      prstmnt.executeUpdate();
      prstmnt.close();
   } catch (SQLException e) {
      e.printStackTrace();
      ok = false;
   }
   return ok;
}

public int ifTestNameExists(String testname) {
   int res = -1;
   try {
      checkConnection();
      PreparedStatement preparedStatement = c.prepareStatement("select " + COL_TEST_ID +
            " from " + TESTS_TABLE_NAME + " where "
            + COL_NAME + " = ?");
      preparedStatement.setString(1, testname);
      ResultSet resultSet = preparedStatement.executeQuery();
      if (resultSet.next()) {
         res = resultSet.getInt(1);
      }
      resultSet.close();
      preparedStatement.close();
   } catch (SQLException e) {
      e.printStackTrace();
   }
   return res;
}



public boolean existTestForUser(int userId, int testId) {
   boolean res = false;
   try {
```

```
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select 1 from " + TESTS_TABLE_NAME + "
where " + COL_USER_ID + " = ? AND " + COL_TEST_ID + " = ? " );
            preparedStatement.setInt(1, userId);
            preparedStatement.setInt(2, testId);
            ResultSet resultSet = preparedStatement.executeQuery();
            res = resultSet.next();
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }


    public int createQuestion(int test_id) {
        int ok;
        String sql = "insert into " + QUEST_TABLE_NAME + " (" + COL_TEST_ID + ", " + COL_TIMER + ") " +
                " values ( ?, ?)";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            prstmnt.setInt(1, test_id);
            prstmnt.setInt(2, 500);
            ok = prstmnt.executeUpdate();
            ResultSet rs = prstmnt.getGeneratedKeys();
            if (rs.next()) {
                ok = rs.getInt(1); // должно вернуть айди вопроса
            }
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
        return ok;
    }

    public int createTest(int userId, String name) {
        name = name.length()>GREAT_CHAR?name.substring(0, GREAT_CHAR):name;
        int ok;
        String sql = "insert into " + TESTS_TABLE_NAME + " (" + COL_USER_ID + ", " + COL_NAME + ") " +
                " values ( ?, ?)";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            prstmnt.setInt(1, userId);
            prstmnt.setString(2, name);
            ok = prstmnt.executeUpdate();
            ResultSet rs = prstmnt.getGeneratedKeys();
            if (rs.next()) {
                ok = rs.getInt(1); // должно вернуть айди теста
            }
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
```

```
    }
    return ok;
}

public boolean addQuestion (int questionId, String question) {
   return addStringColToQuestions(questionId, question, COL_QUESTION);
}

public boolean addAnswerToQuestion(int questionId, String answer) {
    return addStringColToQuestions(questionId, answer, COL_ANSWER);
}

private boolean addStringColToQuestions (int questionId, String string, String column) {
    string = string.length() > GREAT_VARCHAR ? string.substring(0,GREAT_VARCHAR - 1) : string;
    int ok;
    String sql = "UPDATE " + QUEST_TABLE_NAME +
            " SET " + column + " = ?" +
            "  where " + COL_QUESTION_ID + " = ?";
    try {
        checkConnection();
        PreparedStatement prstmnt = c.prepareStatement(sql);
        prstmnt.setString(1, string);
        prstmnt.setInt(2,questionId );
        ok = prstmnt.executeUpdate();
        prstmnt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return ok > 0;
}



public int getUsersTestId(int userId) {
    return getUsersIntCol(userId, COL_TEST_ID);
}
public int getUsersQuestionId(int userId) {
    return getUsersIntCol(userId, COL_QUESTION_ID);
}

private int getUsersIntCol (int userId, String col) {
    int res = -1;
    String sql = "select " + col + " from " + STATES_TABLE_NAME +
            " where " + COL_USER_ID + " = ?";
    try {
        checkConnection();
        PreparedStatement preparedStatement = c.prepareStatement(sql);
        preparedStatement.setInt(1, userId);
        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            res = resultSet.getInt(1);
        }
        resultSet.close();
        preparedStatement.close();
    } catch (SQLException e) {
        e.printStackTrace();
```

```
        }
        return res;
    }


    public boolean addVariant(int questionId, String answer) {
        answer = answer.length()>GREAT_CHAR?answer.substring(0, GREAT_CHAR):answer;
        int ok;
        String sql = "insert into " + VARIANTS_TABLE + " (" + COL_QUESTION_ID + ", " + COL_ANSWER + ") " +
                " values (?, ?)";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            prstmnt.setInt(1, questionId);
            prstmnt.setString(2, answer);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }

    public HashMap<Integer, String> getTestsForUser(int userId) {
        HashMap<Integer, String> res = new HashMap<>();
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement("select " + COL_TEST_ID + ", " + COL_NAME +
                    " from " + TESTS_TABLE_NAME + " where "
                    + COL_USER_ID + " = ?");
            preparedStatement.setInt(1, userId);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                res.put(resultSet.getInt(1), resultSet.getString(2));
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }

    public int createCurrentTest(int userId, int testId) {
        int ok;
        String sql = "insert into " + DONE_TESTS_TABLE_NAME + " (" + COL_USER_ID + ", " + COL_TEST_ID + ", " +
COL_START_TIME + ") " +
                " values ( ?, ?, ?)";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            prstmnt.setInt(1, userId);
            prstmnt.setInt(2, testId);
            prstmnt.setDate(3, new Date(System.currentTimeMillis()));
            ok = prstmnt.executeUpdate();
```

```
            ResultSet rs = prstmnt.getGeneratedKeys();
            if (rs.next()) {
                ok = rs.getInt(1); // должно вернуть айди теста
            }
            rs.close();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
        return ok;
    }



    /**
     * get question id from test that was not answered
     * @param curTestId
     * @return
     */
    public int getQuestionIdOfCurrentTest (int curTestId) {
        int res = -1;
        String sql = "select q." + COL_QUESTION_ID  +
                " from " + QUEST_TABLE_NAME + " q " +
                "left join " + ANSWERS_TABLE_NAME + " a " +
                "on (a." + COL_QUESTION_ID + " = q." + COL_QUESTION_ID + " and a." + COL_CUR_TEST_ID + " = ?) " +
                "where q." + COL_TEST_ID + " = " +
                "(select " + COL_TEST_ID + " from " + DONE_TESTS_TABLE_NAME +
                " where " + COL_CUR_TEST_ID + " = ?) " +
                "and a." + COL_QUESTION_ID + " is null";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, curTestId);
            prstmnt.setInt(2, curTestId);
            ResultSet resultSet = prstmnt.executeQuery();
            if (resultSet.next()) {
                res = resultSet.getInt(1);
            }
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
        return res;
    }

    /**
     * return question text
     * @param questionId
     * @return
     */
    public String getQuestionOfQuestions(int questionId) {
        return getStringOfquestions(questionId, COL_QUESTION);
    }

    public String getAnswerOfQuestions(int questionId) {
```

```
    return getStringOfquestions(questionId, COL_ANSWER);
  }
  private String getStringOfquestions(int questionId, String column) {
    String sql = "select " + column + " from " + QUEST_TABLE_NAME +
        " where " + COL_QUESTION_ID + " = ?";
    String res = "empty";
    try {
      checkConnection();
      PreparedStatement preparedStatement = c.prepareStatement(sql);
      preparedStatement.setInt(1, questionId);
      ResultSet resultSet = preparedStatement.executeQuery();
      if (resultSet.next()) {
        res = resultSet.getString(1);
      }
      resultSet.close();
      preparedStatement.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return res;
  }

  public HashMap<Integer, String>  getAnswersOfDoneTests(int test_id) {
    return getStringsOfTestsTable(test_id, ANSWERS_TABLE_NAME, COL_CUR_TEST_ID);
  }

  public HashMap<Integer, String>  getQuestionsOfTests(int test_id) {
    return getStringsOfTestsTable(test_id, QUEST_TABLE_NAME, COL_TEST_ID);
  }


  private HashMap<Integer, String> getStringsOfTestsTable(int test_id, String tableName, String col) {
    HashMap<Integer, String > res = new HashMap<>();
    try {
      checkConnection();
      PreparedStatement preparedStatement = c.prepareStatement("select " + COL_QUESTION_ID + ", " +
COL_ANSWER +
          " from " + tableName + " where "
          + col + " = ? ");
      preparedStatement.setInt(1, test_id);
      ResultSet resultSet = preparedStatement.executeQuery();
      while (resultSet.next()) {
        res.put(resultSet.getInt(1), resultSet.getString(2));
      }
      resultSet.close();
      preparedStatement.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return res;
  }

  public int getTestIdOfDoneTests(int cur_test_id) {
    int res = -1;
    try {
      checkConnection();
      PreparedStatement preparedStatement = c.prepareStatement("select " + COL_TEST_ID +
```

```
                " from " + DONE_TESTS_TABLE_NAME + " where "
                + COL_CUR_TEST_ID + " = ?");
            preparedStatement.setInt(1, cur_test_id);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                res = resultSet.getInt(1);
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }

    public String getResultsOfTest(int chatId, int testId) {
        String sql = "select u." + COL_NAME + ", d." + COL_MARK + ", d." + COL_PASSED +
        " from " + DONE_TESTS_TABLE_NAME + " d " +
                " join " +  USERS_TABLE_NAME + " u  on (" +
                "d." + COL_USER_ID + " = " + "u." + COL_USER_ID + ")" +
                " where d." + COL_USER_ID + " in (" +
                "select " + COL_USER_ID + " from " + US_GROUP_TABLE_NAME + " where " + COL_CHAT_ID + " = ? and " +
COL_ROLE + " =  ? )" +
                " and " + COL_TEST_ID + " = ? order by " + COL_START_TIME;
        StringBuilder s = new StringBuilder();
        try {
            checkConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, chatId);
            preparedStatement.setString(2, Roles.GROUP_ROLE_STUDENT);
            preparedStatement.setInt(3, testId);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                s.append(resultSet.getString(1));
                s.append(": ");
                s.append(resultSet.getInt(2));
                s.append("% ");
                s.append(resultSet.getInt(3) > 0 ? "passed": "not passed");
                s.append("\n");
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return s.toString();
    }

    public boolean setQuestType(int questionId, String questType) {
        return  updateQuestString(questionId, questType, COL_TYPE);
    }

    public boolean setQuestData(int questionId, String data) {
        return  updateQuestString(questionId, data, COL_DATA);
    }

    public boolean updateQuestString(int questionId, String data, String column) {
```

```
        int ok;
        String sql = "UPDATE " + QUEST_TABLE_NAME + " set " + column + " = ? where " + COL_QUESTION_ID + " = ? ";
        try {
            checkConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setString(1, data);
            prstmnt.setInt(2,questionId );
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok>0;
    }

}


package model.database;

import model.entities.Answer;
import model.entities.Question;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashSet;
import java.util.Set;

import static model.database.DbConstants.ANSWERS_TABLE_NAME;
import static model.database.DbConstants.COL_CUR_TEST_ID;
import static model.database.DbConstants.COL_QUESTION_ID;
import static model.database.DbConstants.COL_START_TIME;
import static model.database.DbConstants.COL_USER_ID;
import static model.database.DbConstants.GREAT_CHAR;
import static model.database.DbConstants.GREAT_VARCHAR;

public class DoneTestAnswersDB {

    private final DatabaseManager db;

    public DoneTestAnswersDB(DatabaseManager db) {
        this.db = db;
    }

    /*
    update answers set answer = '', is_right = '', end_time = sysdate()
    where user_id = 1 and cur_test_id = 1 and question_id = 1
     */
    public boolean updateDoneAnswer(int questionId, int curTestId, String answer, int userId, boolean isRight) {
        int ok = 0;
        answer = answer.length()>GREAT_VARCHAR?answer.substring(0, GREAT_CHAR):answer;
        /* update timer with time difference */
        String sql = "update " + ANSWERS_TABLE_NAME + " set answer = ?, is_right = ?, end_time = sysdate(), timer =
TIMESTAMPDIFF(SECOND, start_time, sysdate())\n" +
                "  where user_id = ? and cur_test_id = ? and question_id = ?";
```

```
    try {
        Connection c = db.getConnection();
        PreparedStatement prstmnt = c.prepareStatement(sql);
        prstmnt.setString(1, answer);
        prstmnt.setInt(2, isRight?1:0);
        prstmnt.setInt(3, userId);
        prstmnt.setInt(4, curTestId);
        prstmnt.setInt(5, questionId);
        ok = prstmnt.executeUpdate();
        prstmnt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return ok > 0;
}
public boolean updateDoneAnswer(int questionId, int curTestId, String answer, int userId) {
    Question question = db.getQuestionDB().getQuestion(questionId);
    boolean isRight = question.getAnswer().toUpperCase().equals(answer.toUpperCase());
    return updateDoneAnswer(questionId, curTestId, answer,userId, isRight);
}

/**
 * without answer, just a start of question
 * @param questionId
 * @param curTestId
 * @param userId
 * @return
 */

// select 1 from answers where user_id = 1 and cur_test_id = 1 and question_id = 1;
public boolean checkExistsAnswer(int questionId, int curTestId, int userId) {
    boolean exists = false;
    String sql = "select 1 from answers where user_id = ? and cur_test_id = ? and question_id = ?";
    try {
        Connection c = db.getConnection();
        PreparedStatement preparedStatement = c.prepareStatement(sql);
        preparedStatement.setInt(1, questionId);
        preparedStatement.setInt(2, curTestId);
        preparedStatement.setInt(3, userId);
        ResultSet resultSet = preparedStatement.executeQuery();
        exists = resultSet.next();
        resultSet.close();
        preparedStatement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return exists;
}

// insert into answers (question_id, cur_test_id, user_id, timer, start_time)
public boolean createDoneAnswer(int questionId, int curTestId, int userId) {
    int ok;
    String sql = "insert into " + ANSWERS_TABLE_NAME + " ("
            + COL_QUESTION_ID + ", "
            + COL_CUR_TEST_ID + ", "
            + COL_USER_ID + ", "
```

```
            + COL_START_TIME  +
            ") " +
            " values (?, ?, ?, sysdate())";
        try {
            Connection c = db.getConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, questionId);
            prstmnt.setInt(2, curTestId);
            prstmnt.setInt(3, userId);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }
    /*select question_id, cur_test_id, user_id, answer, is_right, timer, start_time, end_time from answers;*/
    public Set<Answer> getAnswersByQuestionID(int questionId) {
        Set<Answer> set = new HashSet<>();
        String sql = "select question_id, cur_test_id, user_id, answer, is_right, timer, start_time, end_time from answers "
+
            " where " + COL_QUESTION_ID + " = ?";

        try {
            Connection c = db.getConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, questionId);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                set.add(parseAnswer(resultSet));
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return set;
    }
   // question_id, cur_test_id, user_id, answer, is_right, timer, start_time, end_time
    private Answer parseAnswer(ResultSet resultSet) throws SQLException {
        Answer answer = new Answer(resultSet.getInt(1), resultSet.getInt(2),
                resultSet.getInt(3), resultSet.getString(4),
                resultSet.getInt(5) > 0, resultSet.getInt(6), resultSet.getDate(7), resultSet.getDate(8));
        return answer;    }}

package model.database;

import bot.QuestionTypes;
import model.entities.Question;
import model.entities.Test;
import org.mariadb.jdbc.internal.com.read.resultset.UpdatableColumnDefinition;

import javax.xml.crypto.Data;
import java.awt.image.DataBuffer;
import java.sql.Connection;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

import static model.database.DbConstants.ANSWERS_TABLE_NAME;
import static model.database.DbConstants.COL_ANSWER;
import static model.database.DbConstants.COL_DATA;
import static model.database.DbConstants.COL_FILEPATH;
import static model.database.DbConstants.COL_QUESTION;
import static model.database.DbConstants.COL_QUESTION_ID;
import static model.database.DbConstants.COL_TEST_ID;
import static model.database.DbConstants.COL_TIMER;
import static model.database.DbConstants.COL_TYPE;
import static model.database.DbConstants.GREAT_CHAR;
import static model.database.DbConstants.GREAT_VARCHAR;
import static model.database.DbConstants.QUEST_TABLE_NAME;

public class QuestionTestDB {
    private final DatabaseManager db;

    public QuestionTestDB(DatabaseManager db) {
        this.db = db;
    }

    public Question getQuestion(int questionId) {
        Question q = null;
        String sql = "select " + COL_QUESTION_ID + ", " + COL_TEST_ID + ", " +
                COL_QUESTION + ", " + COL_TYPE + ", " +
                COL_DATA + ", " + COL_ANSWER + ", " + COL_FILEPATH + ", " + COL_TIMER +
                " from " + QUEST_TABLE_NAME +
                " where " + COL_QUESTION_ID + " = ?";

        try {
            Connection c = db.getConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, questionId);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                q = parseQuestion(resultSet);
            }
            resultSet.close();
            preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return q;
    }

    private Question parseQuestion(ResultSet resultSet) throws SQLException {
        QuestionTypes type = QuestionTypes.valueOf(resultSet.getString(4));
        Question kek = new Question(resultSet.getInt(1), resultSet.getInt(2),
                resultSet.getString(3), type, resultSet.getString(5),
                resultSet.getString(6), resultSet.getString(7));
        kek.setTimer(resultSet.getInt(8));
        return kek;
    }
```

```
    public Test getTest(int testId) {
        Test t = new Test(testId);
        String sql = "select " + COL_QUESTION_ID + ", " + COL_TEST_ID + ", " +
                COL_QUESTION_ID + ", " + COL_TYPE + ", " +
                COL_DATA + ", " + COL_ANSWER + ", " + COL_FILEPATH +
                " from " + QUEST_TABLE_NAME +
                " where " + COL_TEST_ID + " = ?";

        try {
            ArrayList<Question> qs = new ArrayList<>();
            Connection c = db.getConnection();
            PreparedStatement preparedStatement = c.prepareStatement(sql);
            preparedStatement.setInt(1, testId);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                qs.add(parseQuestion(resultSet));
            }
            resultSet.close();
            preparedStatement.close();
            t.setQuestions(qs);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return t;
    }

    public boolean updateTimer(int question_id, int timer) {
        int ok = 0;
        String sql = "update " + QUEST_TABLE_NAME + " set timer = ?\n" +
                "  where question_id = ?";
        try {
            Connection c = db.getConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, timer);
            prstmnt.setInt(2, question_id);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }

    public boolean updateDoneAnswer(int questionId, int curTestId, String answer, int userId, boolean isRight) {
        int ok = 0;
        answer = answer.length()>GREAT_VARCHAR?answer.substring(0, GREAT_CHAR):answer;
        /* update timer with time difference */
        String sql = "update " + ANSWERS_TABLE_NAME + " set answer = ?, is_right = ?, end_time = sysdate(), timer =
TIMESTAMPDIFF(SECOND, start_time, sysdate())\n" +
                "  where user_id = ? and cur_test_id = ? and question_id = ?";
        try {
            Connection c = db.getConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setString(1, answer);
```

```
            prstmnt.setInt(2, isRight?1:0);
            prstmnt.setInt(3, userId);
            prstmnt.setInt(4, curTestId);
            prstmnt.setInt(5, questionId);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }

}

package model.database;

import model.entities.User;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import static model.database.DbConstants.COL_CUR_TEST_ID;
import static model.database.DbConstants.COL_END_TIME;
import static model.database.DbConstants.COL_MARK;
import static model.database.DbConstants.COL_PASSED;
import static model.database.DbConstants.DONE_TESTS_TABLE_NAME;

public class UserStateDB {

    private final DatabaseManager db;

    public UserStateDB(DatabaseManager db) {
        this.db = db;
    }

    private User parseUser(ResultSet resultSet) throws SQLException {
        User user = null;
        return user;
    }


    public boolean doneTest (int curTestId, int mark, int passed) {
        int ok;
        String sql = "UPDATE " + DONE_TESTS_TABLE_NAME +
                " SET " + COL_MARK + " = ?, " +
                COL_PASSED + "= ?," +
                COL_END_TIME + " = ? " +
                "  where " + COL_CUR_TEST_ID + " = ?";
        try {
            Connection c = db.getConnection();
            PreparedStatement prstmnt = c.prepareStatement(sql);
            prstmnt.setInt(1, mark);
            prstmnt.setInt(2,passed );
```

```
            prstmnt.setDate(3,  new Date(System.currentTimeMillis()));
            prstmnt.setInt(4, curTestId);
            ok = prstmnt.executeUpdate();
            prstmnt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return ok > 0;
    }}
package model.entities;

import java.sql.Date;

public class Answer {
    private Date start_time;
    private Date end_time;
    private int cur_test_id;
    private int question_id;
    private int user_id;
    private int timer;
    private boolean is_right;
    String answer;

    private User user;
    private DoneTest doneTest;

    public Answer( int question_id, int cur_test_id, int user_id, String answer, boolean is_right, int timer, Date
start_time, Date end_time) {
        this.start_time = start_time;
        this.end_time = end_time;
        this.cur_test_id = cur_test_id;
        this.question_id = question_id;
        this.user_id = user_id;
        this.timer = timer;
        this.is_right = is_right;
        this.answer = answer;
    }

    public Date getStart_time() {
        return start_time;
    }

    public Date getEnd_time() {
        return end_time;
    }

    public int getCur_test_id() {
        return cur_test_id;
    }

    public int getQuestion_id() {
        return question_id;
    }

    public int getUser_id() {
        return user_id;
```

```
    }

    public int getTimer() {
        return timer;
    }

    public boolean is_right() {
        return is_right;
    }

    public String getAnswer() {
        return answer;
    }
}
```

```
package model.entities;

import java.sql.Date;
import java.util.ArrayList;
import java.util.List;

public class DoneTest {
    /**
     * pk_id
     */
    private int cur_test_id;
    private Test test;
    private int test_id;
    private Date start_time;
    private Date end_time;
    private int user_id;
    int mark;
    Boolean passed;

    private User user;
    private List <Answer> answers = new ArrayList<>();


}
```

```
package model.entities;

import bot.QuestionTypes;

public class Question {
    int qiestionId;
    int testId;
    String question;
    QuestionTypes type;
    String data;
    String answer;
    String filePath;
    int timer;

    public Question() {
    }
```

```
    public Question(int qiestionId, int testId, String question, QuestionTypes type, String data, String answer, String
filePath) {
        this.qiestionId = qiestionId;
        this.testId = testId;
        this.question = question;
        this.type = type;
        this.data = data;
        this.answer = answer;
        this.filePath = filePath;
    }

    public int getQiestionId() {
        return qiestionId;
    }

    public void setQiestionId(int qiestionId) {
        this.qiestionId = qiestionId;
    }

    public int getTestId() {       return testId;    }

    public void setTestId(int testId) {       this.testId = testId;    }

    public String getQuestion() {       return question;    }

    public int getTimer() {       return timer;    }

    public void setTimer(int timer) {       this.timer = timer;    }

    public void setQuestion(String question) {       this.question = question;    }

    public QuestionTypes getType() {       return type;    }

    public void setType(QuestionTypes type) {
        this.type = type;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public String getAnswer() {
        return answer;
    }

    public void setAnswer(String answer) {
        this.answer = answer;
    }

    public String getFilePath() {
        return filePath;
    }
```

```
    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }
}

package model.entities;

import java.util.ArrayList;

public class Test {
    int testId;
    ArrayList<Question> questions;

    public Test() {
    }

    public Test(int testId) {
        this.testId = testId;
    }

    public Test(int testId, ArrayList<Question> questions) {
        this.testId = testId;
        this.questions = questions;
    }

    public int getTestId() {
        return testId;
    }

    public void setTestId(int testId) {
        this.testId = testId;
    }

    public ArrayList<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(ArrayList<Question> questions) {
        this.questions = questions;
    }
}

package model.keyboards;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Action", propOrder = {
        "name",
        "id",
        "value",
        "command"
})
public class Action {
```

```
@XmlElement(name = "Name")
protected String name = "";
@XmlElement(name = "Id")
protected String id = "";
@XmlElement(name = "Value")
protected String value = "";
@XmlElement(name = "Command")
protected String command = "";

/**
 * Gets the value of the name property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getName() {
    return name;
}

/**
 * Sets the value of the name property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setName(String value) {
    this.name = value;
}

/**
 * Gets the value of the id property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getId() {
    return id;
}

/**
 * Sets the value of the id property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setId(String value) {
    this.id = value;
}
```

```
    /**
     * Gets the value of the value property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getValue() {
        return value;
    }

    /**
     * Sets the value of the value property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setValue(String value) {
        this.value = value;
    }

    /**
     * Gets the value of the command property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getCommand() {
        return command;
    }

    /**
     * Sets the value of the command property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setCommand(String value) {
        this.command = value;
    }

    }
package model.keyboards;

import org.telegram.telegrambots.meta.api.objects.Update;

public class ActionBuilder {

    private final DocumentMarshaller marshaller;
```

```
    private Action action = new Action();

    public ActionBuilder(DocumentMarshaller marshaller) {
        this.marshaller = marshaller;
    }

    public ActionBuilder setName(String name) {
        action.setName(name);
        return this;
    }

    public ActionBuilder setValue(String name) {
        action.setValue(name);
        return this;
    }

    public String asString() {
        return marshaller.<Action>marshal(action);
    }

    public Action build() {
        return action;
    }

    public Action build(Update update) {
        String data = update.getCallbackQuery().getData();
        if (data == null) {
            return null;
        }

        action = marshaller.<Action>unmarshal(data);

        if (action == null) {
            return null;
        }
        return action;
    }
}

package model.keyboards;

import java.util.Date;

public interface DocumentMarshaller {
    boolean supports(String docTypeCode);

    <T> String marshal(T document);

    <T> T unmarshal(String xml, Date operationDate);

    <T> T unmarshal(String xml);

    <T> T unmarshal(String xml, Class clazz);
}

package model.keyboards;
```

```
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;

public class InlineKeyboardButtonBuilder {

    private final InlineKeyboardButton button;

    public InlineKeyboardButtonBuilder(){
        this.button = new InlineKeyboardButton();
    }


    public InlineKeyboardButtonBuilder setText(String text){
        button.setText(text);
        return this;
    }

    public InlineKeyboardButtonBuilder setCallbackData(String callbackData){
        button.setCallbackData(callbackData);
        return this;
    }

    public InlineKeyboardButton build(){
        return button;
    }
}


package model.keyboards;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import java.io.IOException;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JsonDocumentMarshallerImpl implements DocumentMarshaller {

    private static final Logger log = Logger.getLogger(JsonDocumentMarshallerImpl.class.getSimpleName());
    private JAXBContext context;
    private String docTypeCode;
    private Class<?> clazz;

    public JsonDocumentMarshallerImpl() {
    }

    public JsonDocumentMarshallerImpl(Class clazz, String docTypeCode) {
        this.setDocTypeCode(docTypeCode);
        this.setClazz(clazz);
        init();
    }

    public void init() {
        log.info("got associated class : " + getClazz() + ", docTypeCode == " + getDocTypeCode());
```

```
    try {
        this.setContext(JAXBContext.newInstance(getClazz()));
    } catch (JAXBException e) {
        throw new RuntimeException(e);
    }
}

@Override
public boolean supports(String docTypeCode) {
    return this.getDocTypeCode().equals(docTypeCode);
}

/**
 * Marshalls a document.
 *
 * @param document a document to marshal
 * @return an XMl represenation of the document
 */
@Override
public <T> String marshal(T document) {
    try {
        ObjectMapper objectMapper = new ObjectMapper();
        String v = objectMapper.writeValueAsString(document);
        //writeValueAsString(a, Action.class);
        return v;
    } catch (JsonProcessingException ex) {
        Logger.getLogger(JsonDocumentMarshallerImpl.class.getName()).log(Level.SEVERE, null, ex);
    }
    return null;
}

@Override
public <T> T unmarshal(String xml, Date operationDate) {
    return unmarshal(xml);
}

/**
 * Unmarshalls a document.
 *
 * @param xml an XMl represenation of the document
 */
@Override
public <T> T unmarshal(String xml) {
    return unmarshal(xml, this.clazz);
}

@Override
public <T> T unmarshal(String xml, Class clazz) {
    ObjectMapper objectMapper = new ObjectMapper();
    T s = null;
    try {
        s = (T) objectMapper.readValue(xml, clazz);
    } catch (IOException ex) {
        Logger.getLogger(JsonDocumentMarshallerImpl.class.getName()).log(Level.SEVERE, null, ex);
    }
    return s;
}
```

```
    public JAXBContext getContext() {
        return context;o
    }

    public void setContext(JAXBContext context) {
        this.context = context;
    }

    public String getDocTypeCode() {
        return docTypeCode;
    }

    public void setDocTypeCode(String docTypeCode) {
        this.docTypeCode = docTypeCode;
    }

    public Class<?> getClazz() {
        return clazz;
    }

    public void setClazz(Class<?> clazz) {
        this.clazz = clazz;
    }

}
```

ДОДАТОК Б

Код створення сутностей

```
create table answers
(
        question_id int not null,
        cur_test_id int not null,
        user_id int not null,
        answer varchar(2000) null,
        is_right int default -1 null,
```

```
    timer int null comment 'seconds',
    start_time datetime null,
    end_time datetime null
);

create table done_tests
(
    user_id int not null,
    test_id int not null,
    cur_test_id int auto_increment
        primary key,
    start_time datetime null,
    end_time datetime null,
    mark int null,
    passed int null
);

create table `groups`
(
    chat_id int not null
        primary key,
    name char(100) null
);

create table questions
(
    question_id int auto_increment
        primary key,
    test_id int not null,
    question varchar(2000) null,
    type char(20) null,
    data text null,
    answer varchar(2000) null,
    filepath char(200) null,
    timer int default 300 null
);

create table states
```

```
(
	user_id int not null
		primary key,
	state char(30) null,
	test_id int null,
	question_id int null
);

create table tests
(
	user_id int not null,
	name char(100) null,
	test_id int auto_increment
		primary key
);

create table users
(
	user_id int not null
		primary key,
	name char(100) null,
	nick char(20) null,
	role char(20) null,
	description text null
);

create table users_groups
(
	user_id int not null,
	role char(100) null,
	chat_id int not null
);

create table variants
(
	question_id int not null,
	answer varchar(2000) null
);
```