МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

КАФЕДРА 33

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| старший преподаватель | подпись, дата | К.А.Жиданов |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ОТЧЕТ
# О ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: Технологии и методы программирования

РАБОТУ ВЫПОЛНИЛ

| Студент гр. № 3331 | подпись, дата | А.С.Васильева |
|---|---|---|
| | подпись, дата | инициалы, фамилия |

Санкт-Петербург
2025

## ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Проект представляет собой веб-приложение для управления задачами (ToDo List) с возможностью регистрации пользователей, авторизации и интеграции с Telegram-ботом. Бот позволяет управлять задачами через мессенджер, синхронизируя данные с основной базой данных.

Основные функции:

1) Регистрация и аутентификация пользователей (логин/пароль).
2) CRUD-операции (создание, чтение, обновление, удаление задач).
3) Telegram-бот для управления задачами через чат.
4) Сессии для поддержки авторизованных пользователей.

## ПРИМЕР РАБОТЫ ПРОГРАММЫ

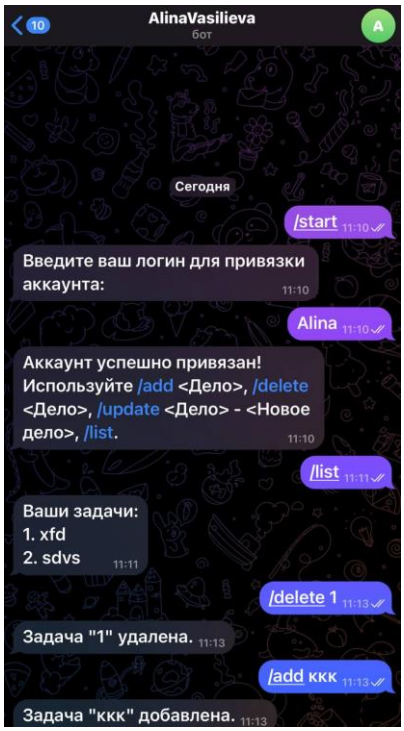


Рисунок 1 – Интеграция телеграмма.

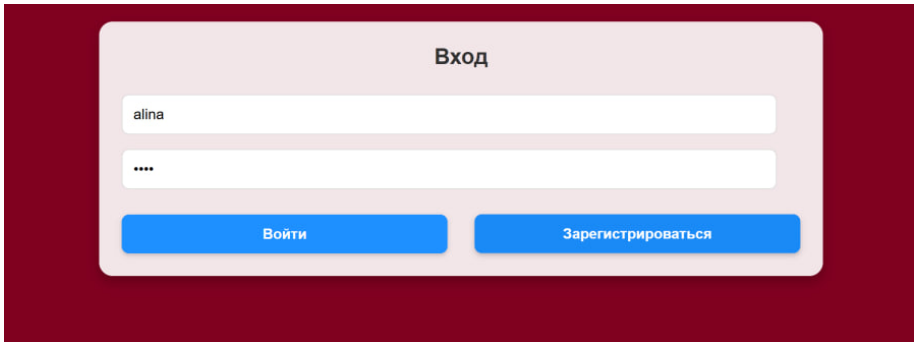


Рисунок 2 – Авторизация.

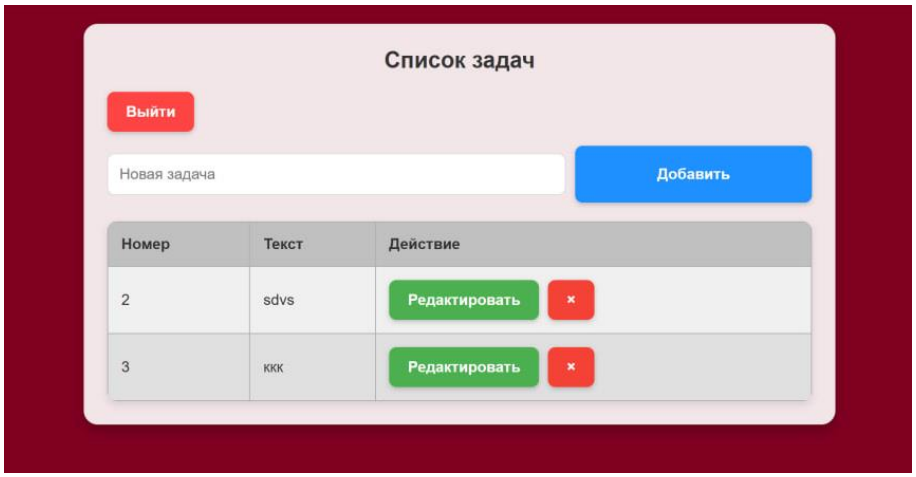


Рисунок 3 – Интерфейс.

## ПРОБЛЕМЫ ПРИ РАБОТЕ С ИИ

### 1.Проблемы с базой данных

**Выявленные проблемы:**

**Инициализация БД в коде (initDatabase):**

При каждом запуске сервера таблица items удалялась (DROP TABLE), что приводило к потере данных.

Не учитывалось, что таблицы могут уже существовать.

**Ошибки подключения к MySQL:**

Если сервер БД недоступен, приложение падало без попытки переподключения.

Некорректные настройки подключения (host, user, password) могли вызывать ошибки.

**Проблемы с FOREIGN KEY:**

При удалении пользователя не учитывалась связь с таблицей items, что могло нарушить целостность данных.

**Решения:**

**Убрана команда DROP TABLE** – теперь таблицы создаются только при первом запуске (CREATE TABLE IF NOT EXISTS).
**Добавлена обработка ошибок подключения** – сервер выводит предупреждение, но продолжает работу.
**Добавлены транзакции** для критических операций (регистрация, привязка Telegram).
**Внешние ключи (FOREIGN KEY) проверяются** перед удалением пользователя.

## 2. Проблемы с недопониманием запросов GPT

**Выявленные проблемы:**

**Неточные формулировки задач:**

Например, запрос "сделай интеграцию с Telegram" не учитывал, что бот должен работать только для привязанных пользователей.

GPT иногда предлагал неоптимальные решения (например, хранение паролей в открытом виде).

**Ошибки в логике бота:**

Первоначально бот не проверял, привязан ли Telegram ID к пользователю.

Команды /delete и /update работали некорректно из-за неправильного парсинга аргументов.

**Решения:**

**Уточнение требований:**

- Четкое описание, что бот должен:

- Проверять привязку аккаунта (/start + логин).
- Работать только с задачами текущего пользователя.

- Добавлена **верификация ввода** (например, проверка trim() для текста задач).

**Доработка кода:**

Исправлен парсинг команд (например, /update <старая задача> - <новая задача>).

Добавлены **проверки прав доступа** (getUserByTelegramId).

## 3. Проблемы с интерфейсом

**Выявленные проблемы:**

1. **Наложение элементов:**

Форма авторизации и список задач отображались одновременно.

После входа страница не обновлялась автоматически.

**Отсутствие обработки ошибок на фронтенде:**

Если сервер возвращал 500 ошибку, интерфейс "зависал".

**Проблемы с Telegram Webhook (если используется HTTPS):**

Бот не получал обновления из-за неправильной конфигурации.

**Решения:**

**Динамическое управление видимостью блоков:**

- После входа authContainer скрывается, todoContainer показывается.
- Добавлен **перехватчик ошибок** (try-catch в JavaScript).

**Улучшение UX:**

- Сообщения об ошибках (showError) теперь исчезают через 3 секунды.
- Добавлена **подсветка полей** при фокусе.

**Фикс для Telegram Webhook:**

- Если используется HTTPS, добавлен express.json() для обработки входящих запросов.

## РАБОТА ПРОЦЕССОВ

### 1.Как работает авторизация

#### 1.1. Регистрация

1. Пользователь вводит **логин и пароль**.
2. Пароль **хешируется** (bcrypt.hash).
3. Данные сохраняются в таблицу users.

**SQL-запрос:**

```
INSERT INTO users (username, password) VALUES (?, ?)
```

#### 1.2. Вход

1. Пользователь вводит логин и пароль.
2. Сервер ищет пользователя в БД и **сравнивает хеши** (bcrypt.compare).
3. Если пароль верный – создается **сессия** (req.session.user).

**Код:**

```
app.post('/login', async (req, res) => {

  const { username, password } = req.body;

  const user = await getUserFromDB(username);

  if (user && await bcrypt.compare(password, user.password)) {
```

```
    req.session.user = { id: user.id, username: user.username };
    res.json({ success: true });
  } else {
    res.status(401).json({ error: "Неверный логин или пароль" });
  }
});
```

**1.3. Выход**

Уничтожается сессия:

```
app.get('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/login');
});
```

## 2. Интеграция с Telegram

**2.1. Привязка аккаунта**

1. Пользователь пишет боту /start.
2. Бот просит ввести **логин из веб-приложения**.
3. Если логин верный – Telegram ID сохраняется в users.telegram_id.

**Код:**

```
bot.onText(/\/start/, async (msg) => {
  const chatId = msg.chat.id;
```

```
  bot.sendMessage(chatId, "Введите ваш логин:");

  bot.once('message', async (msg) => {

    const username = msg.text.trim();

    await linkTelegramId(username, chatId.toString());

  });

});
```

**2.2. Работа с задачами через бота**

- **Добавление:** /add Купить молоко → записывается в БД.
- **Удаление:** /delete 1 → удаляется задача с ID=1.
- **Список:** /list → вывод всех задач пользователя.

**Пример SQL-запроса для /add:**

```
INSERT INTO items (text, user_id) VALUES (?, ?)
```

**Пример SQL-запроса для /list:**

```
SELECT text FROM items WHERE user_id = ?
```

## ВЫВОД ЛАБОРАТОРНОЙ РАБОТЫ

Разработанная и реализованная работа успешно выполняет требования поставленной задачи. Программа обеспечивает корректную интеграцию телеграмма и авторизацию, с возможностью последующих усовершенствований. В работе мы столкнулись с проблемами недопонимания ИИ человеческого запроса, который было необходимо устранить. Решение проблемы получилось достичь с помощью уточняющих вопросов-ответов для ИИ, что дало положительный результат.

## ПРИЛОЖЕНИЕ 1. КОД ПРОГРАММЫ INDEX.JS

```
CONST EXPRESS = REQUIRE('EXPRESS');

CONST MYSQL = REQUIRE('MYSQL2/PROMISE');

CONST SESSION = REQUIRE('EXPRESS-SESSION');

CONST BCRYPT = REQUIRE('BCRYPT');

CONST PATH = REQUIRE('PATH');

CONST FS = REQUIRE('FS').PROMISES;

CONST TELEGRAMBOT = REQUIRE('NODE-TELEGRAM-BOT-API');


CONST APP = EXPRESS();

CONST PORT = 3000;

CONST  TELEGRAM_TOKEN  =  '7994364892:AAHZBPCTW4IYTTHTV7SYYUWIE-JFEMHQGW0'; // ЗАМЕНИТЕ НА ВАШ ТОКЕН

CONST BOT = NEW TELEGRAMBOT(TELEGRAM_TOKEN, { POLLING: TRUE });


CONST DBCONFIG = {
   HOST: 'LOCALHOST',
   USER: 'ROOT',
   PASSWORD: '19122005П',
   DATABASE: 'TODOLIST',
};


APP.USE(EXPRESS.JSON());
APP.USE(SESSION({
   SECRET: 'YOUR-SECRET-KEY',
   RESAVE: FALSE,
```

```
    SAVEUNINITIALIZED: FALSE,
    COOKIE: { SECURE: FALSE } // УСТАНОВИТЕ SECURE: TRUE ДЛЯ HTTPS
}));

ASYNC FUNCTION INITDATABASE() {
    TRY {
        CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION({
            HOST: DBCONFIG.HOST,
            USER: DBCONFIG.USER,
            PASSWORD: DBCONFIG.PASSWORD
        });
        CONSOLE.LOG('CONNECTED TO MYSQL SERVER');

            AWAIT CONNECTION.QUERY('CREATE DATABASE IF NOT EXISTS TODOLIST');
        AWAIT CONNECTION.QUERY('USE TODOLIST');
        CONSOLE.LOG('DATABASE TODOLIST SELECTED');

        AWAIT CONNECTION.QUERY(`
            CREATE TABLE IF NOT EXISTS USERS (
                ID INT AUTO_INCREMENT PRIMARY KEY,
                USERNAME VARCHAR(50) NOT NULL UNIQUE,
                PASSWORD VARCHAR(255) NOT NULL,
                TELEGRAM_ID VARCHAR(50) UNIQUE
            )
        `);
```

```
        CONSOLE.LOG('TABLE USERS CREATED');

        AWAIT CONNECTION.QUERY('DROP TABLE IF EXISTS ITEMS');
        AWAIT CONNECTION.QUERY(`
          CREATE TABLE ITEMS (
            ID INT AUTO_INCREMENT PRIMARY KEY,
            TEXT VARCHAR(255) NOT NULL,
            USER_ID INT,
            FOREIGN KEY (USER_ID) REFERENCES USERS(ID)
          )
        `);
        CONSOLE.LOG('TABLE ITEMS CREATED');

        AWAIT CONNECTION.END();
        CONSOLE.LOG('DATABASE AND TABLES INITIALIZED SUCCESSFULLY');
      } CATCH (ERROR) {
        CONSOLE.ERROR('ERROR INITIALIZING DATABASE:', ERROR);
        THROW ERROR;
      }
    }

    ASYNC FUNCTION RETRIEVELISTITEMS(USERID) {
      TRY {
                                    CONST CONNECTION = AWAIT
MYSQL.CREATECONNECTION(DBCONFIG);
        CONST QUERY = 'SELECT ID, TEXT FROM ITEMS WHERE USER_ID = ?';
```

```
            CONST [ROWS] = AWAIT CONNECTION.EXECUTE(QUERY, [USERID]);
            AWAIT CONNECTION.CLOSE();
            RETURN ROWS;
        } CATCH (ERROR) {
            CONSOLE.ERROR('ОШИБКА ПРИ ПОЛУЧЕНИИ ЭЛЕМЕНТОВ:', ERROR);
            THROW ERROR;
        }
    }

    ASYNC FUNCTION ADDLISTITEM(TEXT, USERID) {
        TRY {
                                    CONST CONNECTION = AWAIT
MYSQL.CREATECONNECTION(DBCONFIG);
            CONST QUERY = 'INSERT INTO ITEMS (TEXT, USER_ID) VALUES (?, ?)';
                CONST [RESULT] = AWAIT CONNECTION.EXECUTE(QUERY, [TEXT,
USERID]);
            AWAIT CONNECTION.CLOSE();
            RETURN { ID: RESULT.INSERTID, TEXT };
        } CATCH (ERROR) {
            CONSOLE.ERROR('ОШИБКА ПРИ ДОБАВЛЕНИИ ЭЛЕМЕНТА:', ERROR);
            THROW ERROR;
        }
    }

    ASYNC FUNCTION DELETELISTITEM(ID, USERID) {
        TRY {
```

```
                CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
                CONST QUERY = 'DELETE FROM ITEMS WHERE ID = ? AND USER_ID = ?';
                CONST [RESULT] = AWAIT CONNECTION.EXECUTE(QUERY, [ID, USERID]);
                AWAIT CONNECTION.CLOSE();
                RETURN RESULT.AFFECTEDROWS > 0;
            } CATCH (ERROR) {
                CONSOLE.ERROR('ОШИБКА ПРИ УДАЛЕНИИ ЭЛЕМЕНТА:', ERROR);
                THROW ERROR;
            }
        }

        ASYNC FUNCTION UPDATELISTITEM(ID, NEWTEXT, USERID) {
            TRY {
                CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
                CONST QUERY = 'UPDATE ITEMS SET TEXT = ? WHERE ID = ? AND USER_ID = ?';
                CONST [RESULT] = AWAIT CONNECTION.EXECUTE(QUERY, [NEWTEXT, ID, USERID]);
                AWAIT CONNECTION.CLOSE();
                RETURN RESULT.AFFECTEDROWS > 0;
            } CATCH (ERROR) {
                CONSOLE.ERROR('ОШИБКА ПРИ ОБНОВЛЕНИИ ЭЛЕМЕНТА:', ERROR);
                THROW ERROR;
            }
```

```
    }

    ASYNC FUNCTION GETUSERBYTELEGRAMID(TELEGRAMID) {
      TRY {
                                    CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
        CONST [ROWS] = AWAIT CONNECTION.EXECUTE('SELECT * FROM USERS WHERE TELEGRAM_ID = ?', [TELEGRAMID]);
        AWAIT CONNECTION.CLOSE();
        RETURN ROWS[0];
      } CATCH (ERROR) {
          CONSOLE.ERROR('ОШИБКА ПРИ ПОЛУЧЕНИИ ПОЛЬЗОВАТЕЛЯ:', ERROR);
        THROW ERROR;
      }
    }

    ASYNC FUNCTION LINKTELEGRAMID(USERNAME, TELEGRAMID) {
      TRY {
                                    CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
        CONST [USERROWS] = AWAIT CONNECTION.EXECUTE('SELECT * FROM USERS WHERE USERNAME = ?', [USERNAME]);
        IF (USERROWS.LENGTH === 0) {
          AWAIT CONNECTION.CLOSE();
          RETURN { SUCCESS: FALSE, MESSAGE: `ПОЛЬЗОВАТЕЛЬ С ЛОГИНОМ "${USERNAME}" НЕ НАЙДЕН. ЗАРЕГИСТРИРУЙТЕСЬ НА САЙТЕ.` };
```

```
            }

             CONST [TELEGRAMROWS] = AWAIT CONNECTION.EXECUTE('SELECT * FROM USERS WHERE TELEGRAM_ID = ?', [TELEGRAMID]);

            IF (TELEGRAMROWS.LENGTH > 0) {

               AWAIT CONNECTION.CLOSE();

                  RETURN { SUCCESS: FALSE, MESSAGE: `ЭТОТ TELEGRAM ID УЖЕ ПРИВЯЗАН К ПОЛЬЗОВАТЕЛЮ "${TELEGRAMROWS[0].USERNAME}".` };

            }

             AWAIT CONNECTION.EXECUTE('UPDATE USERS SET TELEGRAM_ID = ? WHERE USERNAME = ?', [TELEGRAMID, USERNAME]);

            AWAIT CONNECTION.CLOSE();

           CONSOLE.LOG(`TELEGRAM ID ${TELEGRAMID} УСПЕШНО ПРИВЯЗАН К ПОЛЬЗОВАТЕЛЮ ${USERNAME}`);

            RETURN { SUCCESS: TRUE, MESSAGE: 'АККАУНТ УСПЕШНО ПРИВЯЗАН! ИСПОЛЬЗУЙТЕ /ADD <ДЕЛО>, /DELETE <ДЕЛО>, /UPDATE <ДЕЛО> - <НОВОЕ ДЕЛО>, /LIST.' };

         } CATCH (ERROR) {

            CONSOLE.ERROR('ОШИБКА ПРИ ПРИВЯЗКЕ TELEGRAM ID:', ERROR);

               RETURN { SUCCESS: FALSE, MESSAGE: 'ОШИБКА СЕРВЕРА ПРИ ПРИВЯЗКЕ TELEGRAM ID. ПОПРОБУЙТЕ ПОЗЖЕ.' };

         }

      }


      ASYNC FUNCTION GETHTMLROWS(USERID) {

         CONST TODOITEMS = AWAIT RETRIEVELISTITEMS(USERID);

         RETURN TODOITEMS.MAP(ITEM => `

            <TR>
```

```
            <TD>${ITEM.ID}</TD>
            <TD CLASS="TEXT-CELL" DATA-ID="${ITEM.ID}">${ITEM.TEXT}</TD>
            <TD>
                <BUTTON CLASS="EDIT-BTN" DATA-ID="${ITEM.ID}">РЕДАКТИРОВАТЬ</BUTTON>
                <BUTTON CLASS="DELETE-BTN" DATA-ID="${ITEM.ID}">×</BUTTON>
            </TD>
          </TR>
        `).JOIN('');
    }

    // MIDDLEWARE ДЛЯ ПРОВЕРКИ АВТОРИЗАЦИИ
    FUNCTION ISAUTHENTICATED(REQ, RES, NEXT) {
      IF (REQ.SESSION.USER) {
        NEXT();
      } ELSE {
        RES.REDIRECT('/LOGIN');
      }
    }

    APP.GET('/LOGIN', ASYNC (REQ, RES) => {
        CONST HTML = AWAIT FS.READFILE(PATH.JOIN(__DIRNAME, 'INDEX.HTML'), 'UTF8');
      RES.SEND(HTML.REPLACE('{{ROWS}}', ''));
    });
```

```
APP.POST('/LOGIN', ASYNC (REQ, RES) => {
  CONST { USERNAME, PASSWORD } = REQ.BODY;
   IF (!USERNAME || !PASSWORD || USERNAME.TRIM().LENGTH === 0 || PASSWORD.TRIM().LENGTH === 0) {
    RETURN RES.STATUS(400).JSON({ ERROR: 'ЛОГИН И ПАРОЛЬ НЕ МОГУТ БЫТЬ ПУСТЫМИ' });
  }
  TRY {
                                CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
    CONST [ROWS] = AWAIT CONNECTION.EXECUTE('SELECT * FROM USERS WHERE USERNAME = ?', [USERNAME]);
    AWAIT CONNECTION.CLOSE();
    IF (ROWS.LENGTH === 0) {
          RETURN RES.STATUS(401).JSON({ ERROR: 'ПОЛЬЗОВАТЕЛЬ НЕ НАЙДЕН' });
    }
    CONST USER = ROWS[0];
          CONST MATCH = AWAIT BCRYPT.COMPARE(PASSWORD, USER.PASSWORD);
    IF (MATCH) {
      REQ.SESSION.USER = { ID: USER.ID, USERNAME: USER.USERNAME };
      RETURN RES.JSON({ MESSAGE: 'УСПЕШНЫЙ ВХОД' });
    } ELSE {
      RETURN RES.STATUS(401).JSON({ ERROR: 'НЕВЕРНЫЙ ПАРОЛЬ' });
    }
```

```
    } CATCH (ERROR) {
      CONSOLE.ERROR(ERROR);
      RETURN RES.STATUS(500).JSON({ ERROR: 'ОШИБКА СЕРВЕРА' });
    }
  });

  APP.POST('/REGISTER', ASYNC (REQ, RES) => {
    CONST { USERNAME, PASSWORD } = REQ.BODY;
     IF (!USERNAME || !PASSWORD || USERNAME.TRIM().LENGTH === 0 || PASSWORD.TRIM().LENGTH === 0) {
      RETURN RES.STATUS(400).JSON({ ERROR: 'ЛОГИН И ПАРОЛЬ НЕ МОГУТ БЫТЬ ПУСТЫМИ' });
    }
    TRY {
      CONST CONNECTION = AWAIT MYSQL.CREATECONNECTION(DBCONFIG);
      CONST HASHEDPASSWORD = AWAIT BCRYPT.HASH(PASSWORD, 10);
       AWAIT CONNECTION.EXECUTE('INSERT INTO USERS (USERNAME, PASSWORD) VALUES (?, ?)', [USERNAME, HASHEDPASSWORD]);
      AWAIT CONNECTION.CLOSE();
      RETURN RES.JSON({ MESSAGE: 'ПОЛЬЗОВАТЕЛЬ ЗАРЕГИСТРИРОВАН' });
    } CATCH (ERROR) {
      CONSOLE.ERROR(ERROR);
       RETURN RES.STATUS(400).JSON({ ERROR: 'ПОЛЬЗОВАТЕЛЬ УЖЕ СУЩЕСТВУЕТ ИЛИ ОШИБКА СЕРВЕРА' });
    }
  });
```

```
APP.GET('/LOGOUT', (REQ, RES) => {
  REQ.SESSION.DESTROY();
  RES.REDIRECT('/LOGIN');
});

APP.GET('/', ISAUTHENTICATED, ASYNC (REQ, RES) => {
  TRY {
        CONST HTML = AWAIT FS.READFILE(PATH.JOIN(__DIRNAME, 'INDEX.HTML'), 'UTF8');
      CONST PROCESSEDHTML = HTML.REPLACE('{{ROWS}}', AWAIT GETHTMLROWS(REQ.SESSION.USER.ID));
    RES.SEND(PROCESSEDHTML);
  } CATCH (ERR) {
    CONSOLE.ERROR(ERR);
    RES.STATUS(500).SEND('ОШИБКА ЗАГРУЗКИ СТРАНИЦЫ');
  }
});

APP.POST('/ADD', ISAUTHENTICATED, ASYNC (REQ, RES) => {
  CONST { TEXT } = REQ.BODY;
  IF (!TEXT || TYPEOF TEXT !== 'STRING' || TEXT.TRIM() === '') {
        RES.STATUS(400).JSON({ ERROR: 'НЕКОРРЕКТНЫЙ ИЛИ ОТСУТСТВУЮЩИЙ ТЕКСТ' });
    RETURN;
  }
```

```
    TRY {
            CONST NEWITEM = AWAIT ADDLISTITEM(TEXT.TRIM(), REQ.SESSION.USER.ID);
      RES.JSON(NEWITEM);
    } CATCH (ERR) {
      CONSOLE.ERROR(ERR);
      RES.STATUS(500).JSON({ ERROR: 'НЕ УДАЛОСЬ ДОБАВИТЬ ЭЛЕМЕНТ' });
    }
  });

  APP.DELETE('/DELETE', ISAUTHENTICATED, ASYNC (REQ, RES) => {
    CONST ID = REQ.QUERY.ID;
    IF (!ID || ISNAN(ID)) {
            RES.STATUS(400).JSON({ ERROR: 'НЕКОРРЕКТНЫЙ ИЛИ ОТСУТСТВУЮЩИЙ ID' });
      RETURN;
    }
    TRY {
      CONST SUCCESS = AWAIT DELETELISTITEM(ID, REQ.SESSION.USER.ID);
      IF (SUCCESS) {
        RES.JSON({ MESSAGE: 'ЭЛЕМЕНТ УДАЛЕН' });
      } ELSE {
        RES.STATUS(404).JSON({ ERROR: 'ЭЛЕМЕНТ НЕ НАЙДЕН' });
      }
    } CATCH (ERR) {
      CONSOLE.ERROR(ERR);
```

```
        RES.STATUS(500).JSON({ ERROR: 'НЕ УДАЛОСЬ УДАЛИТЬ ЭЛЕМЕНТ' });
      }
    });

    APP.PUT('/UPDATE', ISAUTHENTICATED, ASYNC (REQ, RES) => {
      CONST ID = REQ.QUERY.ID;
      CONST { TEXT } = REQ.BODY;
      IF (!ID || ISNAN(ID) || !TEXT || TYPEOF TEXT !== 'STRING' || TEXT.TRIM() === '')
{
            RES.STATUS(400).JSON({ ERROR: 'НЕКОРРЕКТНЫЙ ИЛИ
ОТСУТСТВУЮЩИЙ ID ИЛИ ТЕКСТ' });
        RETURN;
      }
      TRY {
          CONST SUCCESS = AWAIT UPDATELISTITEM(ID, TEXT.TRIM(),
REQ.SESSION.USER.ID);
        IF (SUCCESS) {
          RES.JSON({ MESSAGE: 'ЭЛЕМЕНТ ОБНОВЛЕН', TEXT });
        } ELSE {
          RES.STATUS(404).JSON({ ERROR: 'ЭЛЕМЕНТ НЕ НАЙДЕН' });
        }
      } CATCH (ERR) {
        CONSOLE.ERROR(ERR);
        RES.STATUS(500).JSON({ ERROR: 'НЕ УДАЛОСЬ ОБНОВИТЬ ЭЛЕМЕНТ' });
      }
    });
```

```
// TELEGRAM BOT
BOT.ONTEXT(/\/START/, ASYNC (MSG) => {
  CONST CHATID = MSG.CHAT.ID;
   BOT.SENDMESSAGE(CHATID, 'ВВЕДИТЕ ВАШ ЛОГИН ДЛЯ ПРИВЯЗКИ АККАУНТА:');
  BOT.ONCE('MESSAGE', ASYNC (MSG) => {
    CONST USERNAME = MSG.TEXT.TRIM();
        CONST RESULT = AWAIT LINKTELEGRAMID(USERNAME, CHATID.TOSTRING());
    BOT.SENDMESSAGE(CHATID, RESULT.MESSAGE);
  });
});

BOT.ONTEXT(/\/ADD (.+)/, ASYNC (MSG, MATCH) => {
  CONST CHATID = MSG.CHAT.ID;
  CONST TEXT = MATCH[1].TRIM();
  CONST USER = AWAIT GETUSERBYTELEGRAMID(CHATID.TOSTRING());
  IF (!USER) {
     BOT.SENDMESSAGE(CHATID, 'ВАШ TELEGRAM ID НЕ ПРИВЯЗАН К АККАУНТУ. ИСПОЛЬЗУЙТЕ /START ДЛЯ ПРИВЯЗКИ.');
    RETURN;
  }
  TRY {
    AWAIT ADDLISTITEM(TEXT, USER.ID);
    BOT.SENDMESSAGE(CHATID, `ЗАДАЧА "${TEXT}" ДОБАВЛЕНА.`);
```

```
    } CATCH (ERROR) {
        BOT.SENDMESSAGE(CHATID, 'ОШИБКА ПРИ ДОБАВЛЕНИИ ЗАДАЧИ.');
    }
});

BOT.ONTEXT(/\/DELETE (.+)/, ASYNC (MSG, MATCH) => {
    CONST CHATID = MSG.CHAT.ID;
    CONST TEXT = MATCH[1].TRIM();
    CONST USER = AWAIT GETUSERBYTELEGRAMID(CHATID.TOSTRING());
    IF (!USER) {
        BOT.SENDMESSAGE(CHATID, 'ВАШ TELEGRAM ID НЕ ПРИВЯЗАН К АККАУНТУ. ИСПОЛЬЗУЙТЕ /START ДЛЯ ПРИВЯЗКИ.');
        RETURN;
    }
    TRY {
        CONST SUCCESS = AWAIT DELETELISTITEM(TEXT, USER.ID);
        IF (SUCCESS) {
            BOT.SENDMESSAGE(CHATID, `ЗАДАЧА "${TEXT}" УДАЛЕНА.`);
        } ELSE {
            BOT.SENDMESSAGE(CHATID, `ЗАДАЧА "${TEXT}" НЕ НАЙДЕНА.`);
        }
    } CATCH (ERROR) {
        BOT.SENDMESSAGE(CHATID, 'ОШИБКА ПРИ УДАЛЕНИИ ЗАДАЧИ.');
    }
});
```

```
BOT.ONTEXT(/\/UPDATE (.+) - (.+)/, ASYNC (MSG, MATCH) => {
  CONST CHATID = MSG.CHAT.ID;
  CONST OLDTEXT = MATCH[1].TRIM();
  CONST NEWTEXT = MATCH[2].TRIM();
  CONST USER = AWAIT GETUSERBYTELEGRAMID(CHATID.TOSTRING());
  IF (!USER) {
    BOT.SENDMESSAGE(CHATID, 'ВАШ TELEGRAM ID НЕ ПРИВЯЗАН К АККАУНТУ. ИСПОЛЬЗУЙТЕ /START ДЛЯ ПРИВЯЗКИ.');
    RETURN;
  }
  TRY {
    CONST SUCCESS = AWAIT UPDATELISTITEM(OLDTEXT, NEWTEXT, USER.ID);
    IF (SUCCESS) {
      BOT.SENDMESSAGE(CHATID, `ЗАДАЧА "${OLDTEXT}" ОБНОВЛЕНА НА "${NEWTEXT}".`);
    } ELSE {
      BOT.SENDMESSAGE(CHATID, `ЗАДАЧА "${OLDTEXT}" НЕ НАЙДЕНА.`);
    }
  } CATCH (ERROR) {
    BOT.SENDMESSAGE(CHATID, 'ОШИБКА ПРИ ОБНОВЛЕНИИ ЗАДАЧИ.');
  }
});

BOT.ONTEXT(/\/LIST/, ASYNC (MSG) => {
  CONST CHATID = MSG.CHAT.ID;
```

```
        CONST USER = AWAIT GETUSERBYTELEGRAMID(CHATID.TOSTRING());
        IF (!USER) {
            BOT.SENDMESSAGE(CHATID, 'ВАШ TELEGRAM ID НЕ ПРИВЯЗАН К АККАУНТУ. ИСПОЛЬЗУЙТЕ /START ДЛЯ ПРИВЯЗКИ.');
          RETURN;
        }
        TRY {
          CONST ITEMS = AWAIT RETRIEVELISTITEMS(USER.ID);
          IF (ITEMS.LENGTH === 0) {
            BOT.SENDMESSAGE(CHATID, 'СПИСОК ЗАДАЧ ПУСТ.');
            RETURN;
          }
                CONST MESSAGE = ITEMS.MAP(ITEM => `${ITEM.ID}. ${ITEM.TEXT}`).JOIN('\N');
          BOT.SENDMESSAGE(CHATID, `ВАШИ ЗАДАЧИ:\N${MESSAGE}`);
        } CATCH (ERROR) {
            BOT.SENDMESSAGE(CHATID, 'ОШИБКА ПРИ ПОЛУЧЕНИИ СПИСКА ЗАДАЧ.');
        }
      });

      INITDATABASE().THEN(() => {
        APP.LISTEN(PORT, () => CONSOLE.LOG(`СЕРВЕР ЗАПУЩЕН НА ПОРТУ ${PORT}`));
      }).CATCH(ERR => {
        CONSOLE.ERROR('FAILED TO INITIALIZE DATABASE AND START SERVER:', ERR);
```

```
    PROCESS.EXIT(1);
});
```

**ПРИЛОЖЕНИЕ 2. КОД ПРОГРАММЫ INDEX.HTML**

```
<!DOCTYPE HTML>
<HTML LANG="RU">
<HEAD>
    <META CHARSET="UTF-8">
     <META NAME="VIEWPORT" CONTENT="WIDTH=DEVICE-WIDTH, INITIAL-SCALE=1.0">
    <TITLE>СПИСОК ЗАДАЧ</TITLE>
    <STYLE>
      BODY {
        FONT-FAMILY: 'ARIAL', SANS-SERIF;
        MAX-WIDTH: 800PX;
        MARGIN: 0 AUTO;
        PADDING: 20PX;
        BACKGROUND-COLOR: #800020; /* БОРДОВЫЙ ФОН */
        COLOR: BLACK;
      }
      .AUTH-CONTAINER, .TODO-CONTAINER {
        MARGIN-BOTTOM: 20PX;
        BACKGROUND-COLOR: RGBA(255, 255, 255, 0.9);
        PADDING: 25PX;
        BORDER-RADIUS: 15PX;
        BOX-SHADOW: 0 6PX 12PX RGBA(0, 0, 0, 0.25);
      }
```

```
H2 {
    COLOR: #333;
    MARGIN-TOP: 0;
    TEXT-ALIGN: CENTER;
    FONT-SIZE: 24PX;
    MARGIN-BOTTOM: 20PX;
}
.INPUT-GROUP {
    MARGIN-BOTTOM: 15PX;
}
INPUT {
    PADDING: 12PX;
    MARGIN: 8PX 0;
    BORDER: 1PX SOLID #DDD;
    BORDER-RADIUS: 8PX;
    WIDTH: CALC(100% - 26PX);
    BOX-SIZING: BORDER-BOX;
    FONT-SIZE: 16PX;
}
.BUTTON-GROUP {
    DISPLAY: FLEX;
    JUSTIFY-CONTENT: SPACE-BETWEEN;
    MARGIN-TOP: 20PX;
}
BUTTON {
```

```
    PADDING: 12PX 20PX;
    BACKGROUND-COLOR: #1E90FF;
    COLOR: WHITE;
    BORDER: NONE;
    BORDER-RADIUS: 8PX;
    CURSOR: POINTER;
    TRANSITION: ALL 0.3S;
    FONT-SIZE: 16PX;
    FONT-WEIGHT: BOLD;
    BOX-SHADOW: 0 2PX 5PX RGBA(0, 0, 0, 0.2);
    WIDTH: 48%; /* ШИРИНА КНОПОК */
}
BUTTON:HOVER {
    BACKGROUND-COLOR: #0066CC;
    TRANSFORM: TRANSLATEY(-2PX);
    BOX-SHADOW: 0 4PX 8PX RGBA(0, 0, 0, 0.3);
}
TABLE {
    WIDTH: 100%;
    BORDER-COLLAPSE: COLLAPSE;
    MARGIN-TOP: 20PX;
    BACKGROUND-COLOR: #E0E0E0;
    BORDER-RADIUS: 10PX;
    OVERFLOW: HIDDEN;
    BOX-SHADOW: 0 4PX 8PX RGBA(0, 0, 0, 0.1);
```

```
}
TH, TD {
   BORDER: 1PX SOLID #B0B0B0;
   PADDING: 14PX;
   TEXT-ALIGN: LEFT;
   COLOR: #333;
}
TH {
   BACKGROUND-COLOR: #C0C0C0;
   FONT-WEIGHT: BOLD;
}
TR:NTH-CHILD(EVEN) {
   BACKGROUND-COLOR: #F0F0F0;
}
.TEXT-CELL {
   CURSOR: POINTER;
}
.ERROR {
   COLOR: RED;
   DISPLAY: NONE;
   MARGIN-TOP: 5PX;
   FONT-WEIGHT: BOLD;
   TEXT-ALIGN: CENTER;
}
.LOGOUT-BTN {
```

```
    BACKGROUND-COLOR: #FF4444;
    WIDTH: AUTO;
    MARGIN-BOTTOM: 15PX;
}
.LOGOUT-BTN:HOVER {
    BACKGROUND-COLOR: #CC0000;
}
.EDIT-BTN {
    BACKGROUND-COLOR: #4CAF50;
    WIDTH: AUTO;
    MARGIN-RIGHT: 5PX;
}
.EDIT-BTN:HOVER {
    BACKGROUND-COLOR: #45A049;
}
.DELETE-BTN {
    BACKGROUND-COLOR: #F44336;
    WIDTH: AUTO;
}
.DELETE-BTN:HOVER {
    BACKGROUND-COLOR: #D32F2F;
}
.TASK-INPUT-GROUP {
    DISPLAY: FLEX;
    GAP: 10PX;
```

```
            MARGIN-BOTTOM: 15PX;
          }
          #TASKINPUT {
            FLEX-GROW: 1;
          }
        </STYLE>
      </HEAD>
      <BODY>
        <DIV CLASS="AUTH-CONTAINER" ID="AUTHCONTAINER">
          <H2>ВХОД</H2>
          <DIV CLASS="INPUT-GROUP">
            <INPUT TYPE="TEXT" ID="USERNAME" PLACEHOLDER="ЛОГИН">
                                    <INPUT TYPE="PASSWORD" ID="PASSWORD"
PLACEHOLDER="ПАРОЛЬ">
          </DIV>
          <DIV CLASS="BUTTON-GROUP">
            <BUTTON ONCLICK="LOGIN()">ВОЙТИ</BUTTON>
           <BUTTON ONCLICK="REGISTER()">ЗАРЕГИСТРИРОВАТЬСЯ</BUTTON>
          </DIV>
          <P ID="AUTHERROR" CLASS="ERROR"></P>
        </DIV>

              <DIV CLASS="TODO-CONTAINER" ID="TODOCONTAINER"
STYLE="DISPLAY: NONE;">
          <H2>СПИСОК ЗАДАЧ</H2>
```

```
                                        <BUTTON CLASS="LOGOUT-BTN" ONCLICK="LOGOUT()">ВЫЙТИ</BUTTON>
            <DIV CLASS="TASK-INPUT-GROUP">
                <INPUT TYPE="TEXT" ID="TASKINPUT" PLACEHOLDER="НОВАЯ ЗАДАЧА">
              <BUTTON ONCLICK="ADDTASK()">ДОБАВИТЬ</BUTTON>
            </DIV>
            <P ID="TASKERROR" CLASS="ERROR"></P>
            <TABLE ID="TASKTABLE">
              <TR>
                <TH>НОМЕР</TH>
                <TH>ТЕКСТ</TH>
                <TH>ДЕЙСТВИЕ</TH>
              </TR>
              {{ROWS}}
            </TABLE>
          </DIV>

          <SCRIPT>
            ASYNC FUNCTION SHOWERROR(ELEMENTID, MESSAGE) {
                                        CONST ERRORELEMENT = DOCUMENT.GETELEMENTBYID(ELEMENTID);
              ERRORELEMENT.TEXTCONTENT = MESSAGE;
              ERRORELEMENT.STYLE.DISPLAY = 'BLOCK';
             SETTIMEOUT(() => { ERRORELEMENT.STYLE.DISPLAY = 'NONE'; }, 3000);
            }
```

```
        ASYNC FUNCTION LOGIN() {
                                                CONST USERNAME =
DOCUMENT.GETELEMENTBYID('USERNAME').VALUE.TRIM();
                                                CONST PASSWORD =
DOCUMENT.GETELEMENTBYID('PASSWORD').VALUE.TRIM();
            IF (USERNAME.LENGTH === 0 || PASSWORD.LENGTH === 0) {
                SHOWERROR('AUTHERROR', 'ЛОГИН И ПАРОЛЬ НЕ МОГУТ БЫТЬ
ПУСТЫМИ');
                RETURN;
            }
            TRY {
                CONST RESPONSE = AWAIT FETCH('/LOGIN', {
                    METHOD: 'POST',
                    HEADERS: { 'CONTENT-TYPE': 'APPLICATION/JSON' },
                    BODY: JSON.STRINGIFY({ USERNAME, PASSWORD })
                });
                CONST DATA = AWAIT RESPONSE.JSON();
                IF (RESPONSE.OK) {
                    DOCUMENT.GETELEMENTBYID('AUTHCONTAINER').STYLE.DISPL
AY = 'NONE';
                    DOCUMENT.GETELEMENTBYID('TODOCONTAINER').STYLE.DISPL
AY = 'BLOCK';
                    LOADTASKS();
                } ELSE {
                    SHOWERROR('AUTHERROR', DATA.ERROR || 'ОШИБКА ВХОДА');
                }
```

```
    } CATCH (ERR) {
        SHOWERROR('AUTHERROR', 'ОШИБКА СЕРВЕРА');
    }
}

ASYNC FUNCTION REGISTER() {
                                    CONST USERNAME = DOCUMENT.GETELEMENTBYID('USERNAME').VALUE.TRIM();
                                    CONST PASSWORD = DOCUMENT.GETELEMENTBYID('PASSWORD').VALUE.TRIM();
    IF (USERNAME.LENGTH === 0 || PASSWORD.LENGTH === 0) {
        SHOWERROR('AUTHERROR', 'ЛОГИН И ПАРОЛЬ НЕ МОГУТ БЫТЬ ПУСТЫМИ');
        RETURN;
    }
    TRY {
        CONST RESPONSE = AWAIT FETCH('/REGISTER', {
            METHOD: 'POST',
            HEADERS: { 'CONTENT-TYPE': 'APPLICATION/JSON' },
            BODY: JSON.STRINGIFY({ USERNAME, PASSWORD })
        });
        CONST DATA = AWAIT RESPONSE.JSON();
        IF (RESPONSE.OK) {
                SHOWERROR('AUTHERROR', 'РЕГИСТРАЦИЯ УСПЕШНА, ВОЙДИТЕ');
        } ELSE {
```

```
                SHOWERROR('AUTHERROR', DATA.ERROR || 'ОШИБКА РЕГИСТРАЦИИ');
            }
        } CATCH (ERR) {
            SHOWERROR('AUTHERROR', 'ОШИБКА СЕРВЕРА');
        }
    }

    ASYNC FUNCTION LOGOUT() {
        TRY {
            AWAIT FETCH('/LOGOUT');
            DOCUMENT.GETELEMENTBYID('TODOCONTAINER').STYLE.DISPLAY = 'NONE';
            DOCUMENT.GETELEMENTBYID('AUTHCONTAINER').STYLE.DISPLAY = 'BLOCK';
            DOCUMENT.GETELEMENTBYID('USERNAME').VALUE = '';
            DOCUMENT.GETELEMENTBYID('PASSWORD').VALUE = '';
        } CATCH (ERR) {
            SHOWERROR('TASKERROR', 'ОШИБКА ВЫХОДА');
        }
    }

    ASYNC FUNCTION LOADTASKS() {
        TRY {
            CONST RESPONSE = AWAIT FETCH('/');
            CONST HTML = AWAIT RESPONSE.TEXT();
```

```
            CONST PARSER = NEW DOMPARSER();
            CONST DOC = PARSER.PARSEFROMSTRING(HTML, 'TEXT/HTML');
            CONST TABLE = DOC.QUERYSELECTOR('#TASKTABLE');
                    DOCUMENT.QUERYSELECTOR('#TASKTABLE').INNERHTML =
TABLE.INNERHTML;
        } CATCH (ERR) {
            SHOWERROR('TASKERROR', 'ОШИБКА ЗАГРУЗКИ ЗАДАЧ');
        }
    }

    ASYNC FUNCTION ADDTASK() {
                                                CONST TEXT =
DOCUMENT.GETELEMENTBYID('TASKINPUT').VALUE.TRIM();
        IF (!TEXT) {
            SHOWERROR('TASKERROR', 'ВВЕДИТЕ ТЕКСТ ЗАДАЧИ');
            RETURN;
        }
        TRY {
            CONST RESPONSE = AWAIT FETCH('/ADD', {
                METHOD: 'POST',
                HEADERS: { 'CONTENT-TYPE': 'APPLICATION/JSON' },
                BODY: JSON.STRINGIFY({ TEXT })
            });
            CONST DATA = AWAIT RESPONSE.JSON();
            IF (RESPONSE.OK) {
                DOCUMENT.GETELEMENTBYID('TASKINPUT').VALUE = '';
```

```
                CONST TABLE = DOCUMENT.GETELEMENTBYID('TASKTABLE');
                CONST ROW = TABLE.INSERTROW(-1);
                ROW.INNERHTML = `
                  <TD>${DATA.ID}</TD>
                                                    <TD CLASS="TEXT-CELL" DATA-ID="${DATA.ID}">${DATA.TEXT}</TD>
                  <TD>
                                                  <BUTTON CLASS="EDIT-BTN" DATA-ID="${DATA.ID}">РЕДАКТИРОВАТЬ</BUTTON>
                                                <BUTTON CLASS="DELETE-BTN" DATA-ID="${DATA.ID}">×</BUTTON>
                  </TD>
                `;
                ATTACHEVENTLISTENERS();
              } ELSE {
                        SHOWERROR('TASKERROR', DATA.ERROR || 'ОШИБКА ДОБАВЛЕНИЯ');
              }
            } CATCH (ERR) {
              SHOWERROR('TASKERROR', 'ОШИБКА СЕРВЕРА');
            }
          }

          ASYNC FUNCTION DELETETASK(ID) {
            TRY {
              CONST RESPONSE = AWAIT FETCH(`/DELETE?ID=${ID}`, {
```

```
            METHOD: 'DELETE'
        });
        CONST DATA = AWAIT RESPONSE.JSON();
        IF (RESPONSE.OK) {
                                    DOCUMENT.QUERYSELECTOR(`TR    TD[DATA-
ID="${ID}"]`).PARENTELEMENT.REMOVE();
        } ELSE {
          SHOWERROR('TASKERROR', DATA.ERROR || 'ОШИБКА УДАЛЕНИЯ');
        }
      } CATCH (ERR) {
        SHOWERROR('TASKERROR', 'ОШИБКА СЕРВЕРА');
      }
    }

    ASYNC FUNCTION EDITTASK(ID, TEXTCELL) {
        CONST NEWTEXT = PROMPT('ВВЕДИТЕ НОВЫЙ ТЕКСТ ЗАДАЧИ:',
TEXTCELL.TEXTCONTENT);
      IF (NEWTEXT === NULL || NEWTEXT.TRIM() === '') RETURN;
      TRY {
        CONST RESPONSE = AWAIT FETCH(`/UPDATE?ID=${ID}`, {
          METHOD: 'PUT',
          HEADERS: { 'CONTENT-TYPE': 'APPLICATION/JSON' },
          BODY: JSON.STRINGIFY({ TEXT: NEWTEXT.TRIM() })
        });
        CONST DATA = AWAIT RESPONSE.JSON();
        IF (RESPONSE.OK) {
```

```
                TEXTCELL.TEXTCONTENT = DATA.TEXT;
            } ELSE {
                SHOWERROR('TASKERROR', DATA.ERROR || 'ОШИБКА РЕДАКТИРОВАНИЯ');
            }
        } CATCH (ERR) {
            SHOWERROR('TASKERROR', 'ОШИБКА СЕРВЕРА');
        }
    }

    FUNCTION ATTACHEVENTLISTENERS() {
        DOCUMENT.QUERYSELECTORALL('.DELETE-BTN').FOREACH(BUTTON => {
            BUTTON.ONCLICK = () => DELETETASK(BUTTON.DATASET.ID);
        });
        DOCUMENT.QUERYSELECTORALL('.EDIT-BTN').FOREACH(BUTTON => {
            BUTTON.ONCLICK = () => {
                CONST TEXTCELL = DOCUMENT.QUERYSELECTOR(`TD[DATA-ID="${BUTTON.DATASET.ID}"]`);
                EDITTASK(BUTTON.DATASET.ID, TEXTCELL);
            };
        });
    }

    DOCUMENT.ADDEVENTLISTENER('DOMCONTENTLOADED', () => {
```

```
        ATTACHEVENTLISTENERS();
    });
  </SCRIPT>
</BODY>
</HTML>
```

## ПРИЛОЖЕНИЕ 3. КОД ПРОГРАММЫ DB.SQL

```
CREATE DATABASE IF NOT EXISTS TODOLIST;
USE TODOLIST;

CREATE TABLE IF NOT EXISTS USERS (
  ID INT AUTO_INCREMENT PRIMARY KEY,
  USERNAME VARCHAR(50) NOT NULL UNIQUE,
  PASSWORD VARCHAR(255) NOT NULL,
  TELEGRAM_ID VARCHAR(50) UNIQUE
);

CREATE TABLE IF NOT EXISTS ITEMS (
  ID INT AUTO_INCREMENT PRIMARY KEY,
  TEXT VARCHAR(255) NOT NULL,
  USER_ID INT,
  FOREIGN KEY (USER_ID) REFERENCES USERS(ID)
);
```