Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

(Университет ИТМО)

Факультет Инфокоммуникационных технологий

Образовательная программа Программирование в инфокоммуникационных системах

О Т Ч Е Т

о практике производственной, технологической

Тема задания: Разработка Telegram-бота для управления платформой Proxmox

Обучающийся Лешков Роман Сергеевич K34212

Согласовано:
Руководитель практики от университета: Самохин Никита Юрьевич

Практика пройдена с оценкой _____

Дата _____

Санкт-Петербург
2023

Оглавление

Введение

Для проектирования сети и реализации веб-сервисов зачастую используются разные подходы и технологии. Одними из самых используемых технологий являются виртуализация и контейнеризация, суть которых заключается в абстрагировании от аппаратной реализации, таким образом при запуске нескольких виртуальных машин или контейнеров обеспечивается логическая изоляция сервисов, процессов и задач от других виртуальных машин и контейнеров.

Proxmox VE – это платформа для управления виртуализацией, имеющая открытый исходный код. Платформа дает возможность управлять виртуальными машинами, контейнерами, объединять их в кластеры, настраивать хранилища и сети. Платформа объединяет виртуализацию KVM и контейнеризацию LXC. При этом реализована в виде серверной части с Rest API и интегрированного веб-интерфейса.

Proxmox VE позволяет оптимизировать существующие ресурсы и повысить эффективность выполнения приложений Linux и Windows, и динамически расширять вычислительные ресурсы и хранилища.

Telegram – кроссплатформенный система с функциями мгновенного обмена текстовыми, голосовыми и видеосообщениями. В Telegram поддерживается создание небольших приложений, которые способны выполнять разнообразные задачи – Telegram-боты.

Telegram-боты подключены к серверу владельца, который обрабатывает входящие запросы от пользователей. Telegram выступает в роли клиентской части приложения, размещенного на сервере. При этом соединение Telegram-сервера с сервером разработчика происходит через https соединение, а для управления Telegram-ботом используется Telegram Bot API.

Golang – язык программирования с открытым исходным кодом, который поддерживает Google. Имеет широкое применение в облачных и сетевых сервисах, для веб-разработки, консольных приложений, в сфере DevOps и SRE.

В данной работе описывается процесс разработки Telegram-бота, который по предоставленным пользователем данным осуществляет управление платформой Proxmox VE, то есть меню Telegram-бота будет использовано в качестве пользовательского интерфейса платформы Proxmox VE.

Данная работа является актуальной, так как виртуализация присутствует на каждом этапе разработки программного обеспечения, и возможность управлять платформой для виртуализации удаленно даже не имея белого IP-адреса на сервере может сократить время реагирования на запросы клиентов или пользователей.

Целью практической работы является разработка Telegram-бота осуществляющего удаленное управление сервером с платформой Proxmox VE.

Для достижения поставленной цели необходимо решить следующие задачи:

- Анализ документации Telegram Bot API
- Анализ документации Proxmox VE API
- Разработка пользовательского интерфейса для Telegram-чата
- Разработка системы действий на основе нажатия кнопок в Telegram-чате
- Написание запросов для Telegram Bot API
- Написание запросов для Proxmox VE API
- Тестирование разработанного Telegram-бота

Анализ документации

В данном разделе будут описаны основные шаги для использования API, и также структуры и методы, которые используются для отправки запросов к Telegram Bot и Proxmox.

Telegram Bot API

Вся документация для Telegram Bot API располагается на официальном сайте Telegram[1]. Начало работы с ботом начинается с регистрации бота. Регистрация происходит через бота @BotFather: запрашиваешь создание нового бота командой “/newBot”, после разработчиком вводится имя бота для управления им, в случае этой работы: “MyProxmox_bot”, после этого имя бота в чате будет отображаться как “MyProxmoxBot”. После имени запрашивается тэг бота, в данной работе “RLeshProxmoxBot”, то есть, чтобы найти этого бота в Telegram нужно искать @RLeshProxmoxBot.

После ввода тэга @BotFather присылает сообщение содержащие токен созданного бота. Именно благодаря токену осуществляется управления ботом: все запросы к боту осуществляются по следующему шаблону: https://api.telegram.org/bot<token>/METHOD_NAME.

Для получения обновлений от Telegram-бота на сервер используется метод “getUpdates”, который возвращает массив объектов типа “Update” (Рисунок 1). Для дальнейшей работы выбраны следующие поля “Update”: “update_id” – id обновления, “message” – новое входящие сообщение, “callback_query” – оповещение об использовании интерактивной клавиатуры. В свою очередь поля “message”,”callback_query” имеют свои поля. У типа “Message”: “message_id” – id сообщения, “from” – id и имя пользователя, “chat” – id чата, “text” – текст сообщения, “reply_markup” – прикрепленная интерактивная

клавиатура. У типа “CallbackQuery”: id” – id оповещения, “from” – id и имя пользователя, “Data” – текст, привязанный к кнопке, и поле “message” – сообщение, к которому прикреплена кнопка.

Рисунок 1 – Структуры для получения обновлений

Модуль клавиатуры представляет из себя таблицу кнопок, которая реализована как массив массивов кнопок, в свою очередь имеет поля: “text” – текст, отображающийся на кнопки и “callback_data” – данные, которые придут на сервер после нажатия на кнопку (Рисунок 2).

Рисунок 2 – Структуры для реализации клавиатуры

Поле “message” присутствует в ответе только в случае, если пришло новое сообщение, а поле “callback_query” только в случае нажатия на кнопку.

Для ответа пользователю выбраны три метода “sendMessage”, “editMessageText” и “answerCallbackQuery”. Первый используется для отправки сообщение в чат, для этого нужно передать параметры: id чата, текстовое сообщение и клавиатуру, второй метод для редактирования текста сообщения бота: требует ту же параметры, что и “sendMessage”, и id сообщения для редактирования. Третий метод нужен для того, чтобы дать обратную связь на нажатие кнопки, и для него указываются id оповещения и всплывающий текст.

Proxmox VE API

Документация Proxmox VE API расположена на официальном сайте Proxmox VE [2]. Важно отметить, что структура этого API построена в виде JSON файла, при этом ответ

будет зависеть от указанного в запросе http-метода. В итоге, все запросы имеют шаблон: <http-метод> https://<server:port>/api2/json/<resource>.

В отличие от Telegram API для использования Proxmox VE API, нужно проходить процесс авторизации: для этого отправляется post-запрос на ресурс “access/ticket” с прикрепленными логином и паролем, в ответ пользователь получает структуру, содержащую поля: “CSRFPreventionToken” – токен, и “ticket” – тикет. Поле тикет используется для авторизации: в последующих запросах оно указывается в cookie запроса с ключом: “PVEAuthCookie”. После указания тикета пользователь считается авторизованным и ему становятся доступны get-запросы согласно уровню доступа пользователя. Для post-, put-, delete-запросов нужно в запросе добавить заголовок “CSRFPreventionToken” с указание токена, с указанием этого заголовка добавляется возможность использовать изменяющие http-методы для ресурсов, доступ к которым есть у пользователя.

Разработка серверной части

Разработка пользовательского интерфейса для Telegram-чата

В качестве пользовательского меню в Telegram-чате выступает текстовое сообщение с прикрепленным модулем клавиатуры.

Использование telegram-бота начинается с нажатия кнопки “Старт” в чате бота, при нажатии которой в чат боту отправляется сообщение с текстом “/start”. После этого должно появиться стартовое меню.

В стартовом меню нужно будет указать данные пользователя: логин, пароль и сервер с Proxmox VE, это меню получило название “userDataMenu”. После ввода данных или нажатия кнопки “Главное меню”, должен произойти переход в главное меню – “mainMenu”. В главном меню должны кнопка для изменения и просмотра текущих данных – переход в userDataMenu, а также кнопка для перехода взаимодействия с сервером – actionMenu. Далее во всех меню должны быть кнопки “Назад” для возвращения в предыдущее меню и “Главное меню” для перехода в mainMenu. В actionMenu начинаются запросы данных от Proxmox VE. На момент написания этого отчета созданы меню: nodesMenu – список узлов в виде клавиатуры, nodeMenu – список действий в виде кнопок над выбранным узлом, включая переход nodeStatusMenu и lxcsMenu. В nodeStatusMenu кнопки для проверки статуса узла, для выключения и перезагрузки узла. В lxcsMenu представлен список контейнеров, lxcMenu – меню для действий с lxc-контейнерами, lxcStatusMenu – меню для просмотра и изменений статуса контейнера – текущий статус, перезагрузить, выключить и запустить контейнер. На Рисунке 3 показаны макеты перечисленных меню.

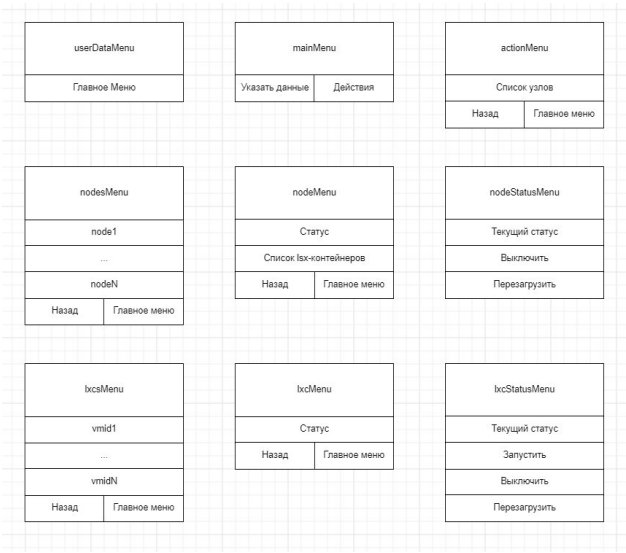

Рисунок 3 – Макеты реализованных меню

Для реализации меню на языке программирования Go создана структура “menu” (Рисунок 4) с двумя полями: “text” – текст и “keyboard” – клавиатура. В поле текста указывается дополнительная информация, в поле клавиатуры указывается массив массивов кнопок. В поле “CallbackData” кнопок указывается команда для вызова меню или путь выбранного ресурса для Proxmox VE API.

```go
type menu struct {
    keyboard [][]InlineKeyboardButton
    text string
}
```

Рисунок 4 – Структура для меню

Меню userDataMenu, mainMenu, actionMenu не зависят от входных данных, поэтому реализованы в виде переменной типа “menu” (Рисунок 5).

```go
var userDataMenu menu = menu{
    text: "Укажите логин, пароль и сервер\n" +
        "В формате:\n" +
        "login: <login>\n" +
        "password: <password>\n" +
        "server: <serverIP:port>\n",
    keyboard: [][]InlineKeyboardButton{
        {
            InlineKeyboardButton{Text: "Главное меню", CallbackData: "/mainMenu"},
        },
    },
}
```

```go
var mainMenu menu = menu{
    text: "Главное меню",
    keyboard: [][]InlineKeyboardButton{
        {
            InlineKeyboardButton{Text: "Изменить данные",
                CallbackData: "/userDataMenu"},
            InlineKeyboardButton{Text: "Действия",
                CallbackData: "/actionMenu"},
        },
    },
}
```

```go
var actionMenu menu = menu{
    text: "Действия",
    keyboard: [][]InlineKeyboardButton{
        {
            InlineKeyboardButton{Text: "Список узлов",
                CallbackData: "/nodes"},
        },
        {
            InlineKeyboardButton{Text: "Назад",
                CallbackData: "/mainMenu"},
            InlineKeyboardButton{Text: "Главное меню",
                CallbackData: "/mainMenu"},
        },
    },
}
```

Рисунок 5 – Меню, не зависящие от входных данных

Остальные меню реализованы в виде функций, которые получают входные данные, строят по ним меню и возвращают его в качестве ответа (Рисунок 6).

```
func nodesMenu(nodes GetNodes) menu{
    menu := menu{text: "Список узлов"}
    for _, node := range nodes.Nodes{
        menu.keyboard = append(
            menu.keyboard,
                []InlineKeyboardButton{
                    InlineKeyboardButton{Text: node.Node,
                        CallbackData: "/nodes/" + node.Node},
                },
        )
    }
    menu.keyboard = append(
        menu.keyboard,
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Действия",
                CallbackData: "/actionMenu"},
            InlineKeyboardButton{Text: "Главное меню",
                CallbackData: "/mainMenu"},
        },
    )
    return menu
}
```

```
func nodeMenu(path string) menu{
    menu := menu{text: "Опции узла " + path}
    menu.keyboard = append(
        menu.keyboard,
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Список lxc-контейнеров",
                CallbackData: path + "/lxc"},
        },
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Статус узла",
                CallbackData: path + "/status"},
        },
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Назад", CallbackData:
                path[:strings.LastIndex(path, substr: "/")]},
            InlineKeyboardButton{Text: "Главное меню",
                CallbackData: "/mainMenu"},
        },
    )
    return menu
}
```

```
func nodeStatusMenu(path string) menu{
    menu := menu{text: "Действия над статусом " +
        path[:strings.LastIndex(path, substr: "/")]}
    menu.keyboard = append(
        menu.keyboard,
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Текущий статус",
                CallbackData: path + "/current"},
        },
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Перезагрузить",
                CallbackData: path + "/reboot"},
        },
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Выключить",
                CallbackData: path + "/shutdown"},
        },
        []InlineKeyboardButton{
            InlineKeyboardButton{Text: "Назад", CallbackData:
            path[:strings.LastIndex(path, substr: "/")]},
            InlineKeyboardButton{Text: "Главное меню",
                CallbackData: "/mainMenu"},
        },
    )
    return menu
}
```

Рисунок 6 – Реализация меню, зависящих от входных данных

Разработка системы действий на основе нажатия кнопок в Telegram-чате

При нажатии кнопок сервер получает “Update”, в котором через поле “callback_query” передан текст указанный за кнопкой. В этом тексте прописаны команды, на которые сервер должен реагировать.

Команды в написанных меню делятся на два типа: вызов меню, не зависящего от пользовательских данных, и вызов меню, зависящих от пользовательских данных.

Для первой группы меню достаточно сравнить команду и имена этих меню, и передать меню через post-запрос.

У второй группы меню в команде могут содержаться пользовательские названия, например узлов, знать которые серверная часть не может. Так что есть два варианта – это передавать кнопкой структуру, содержащую путь и шаблон пути, или написать функцию, определяющую шаблон запроса. От первого варианта пришлось отказаться, так как поле

“callback_data” кнопки может содержать только 64 байта, чего не хватает для некоторых ресурсов. Поэтому была написана функция, определяющая шаблон запроса. Например, если сервер получил команду “/nodes/nodeForDB/lxc/758”, функция должна передать шаблон “/nodes/node/lxc/vmid”.

Суть этой функции в разделении пути запроса на части и сравнением с деревом ресурсов Proxmox VE API.

После получения шаблона шаблон сопоставляется с командой: в зависимости от команды вызывается http-запрос с указанием пути запроса, а полученный ответ используется как аргумент в формировании следующего меню.

Написание запросов для Telegram Bot API

Для получения обновления от Telegram-бота написана функция “getUpdates” (Рисунок 7), в аргументах которой указываются url бота и параметр “offset”. В методе отправляется get-запрос по url, с открытым параметром “offset”. Параметр “offset”, позволяет получать не все обновления на сервере, а только те, чей “update_id” выше параметра значения “offset”.

```
func getUpdates(botUrl string, offset int) ([]Update, error){
    resp, err := http.Get(botUrl + "/getUpdates" + "?offset=" + strconv.Itoa(offset))
    if err != nil{ nil, err }
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil{ nil, err }
    var response Response
    if err := json.Unmarshal(body, &response); err != nil{ nil, err }
    return response.Result, nil
}
```

Рисунок 7 – Функция для получения обновлений от Telegram-бота

Далее для каждого обновления вызываются функции “respondChat”, “editMessage”. Первый отвечает на новые сообщения, используя метод “sendMessage” (Рисунок 8).

```
func respondChat(botUrl string, update Update) error {
    if update.Message.Chat.Id == 0{ nil }
    var botMessage BotMessage
    botMessage.ChatId = update.Message.Chat.Id
    switch update.Message.Text {
    case "/start":
        botMessage.Text = userDataMenu.text
        botMessage.ReplyMarkup.InlineKeyboard = userDataMenu.keyboard
    default:
        userData, err := strToUserData(update.Message.Text)
        if err != nil{
            botMessage.Text = "Неверный формат пользовательских данных\n" + userDataMenu.text
            botMessage.ReplyMarkup.InlineKeyboard = userDataMenu.keyboard
        }else {
            if err := createUserData(update.Message.From.Id, userData); err != nil{
                log.Println( v...: "Error in func createUserData(): ", err)
            }
            botMessage.Text = mainMenu.text + "\nДанные внесены.\n" + userData.String()
            botMessage.ReplyMarkup.InlineKeyboard = mainMenu.keyboard
        }
    }
    buf, err := json.Marshal(botMessage)
    if err != nil{ err }
    _, err = http.Post(botUrl + "/sendMessage",  contentType: "application/json", bytes.NewBuffer(buf))
    if err != nil{ err }
    return nil
}
```

Рисунок 8 – Функция отправки сообщения в Telegram-бота

Второй используется, если пользователь пользуется кнопками, и использует методы “editMessage” и “answerCallbackQuery” (Рисунок 9).

Рисунок 9 – Функция редактирования сообщения в Telegram-бота

Написание запросов для Proxmox VE API

В Proxmox VE API есть авторизация, поэтому пишется отдельная функция на запрос тикета и токена пользователя, которая возвращает структуру с тикетом и токеном пользователя (Рисунок 10).

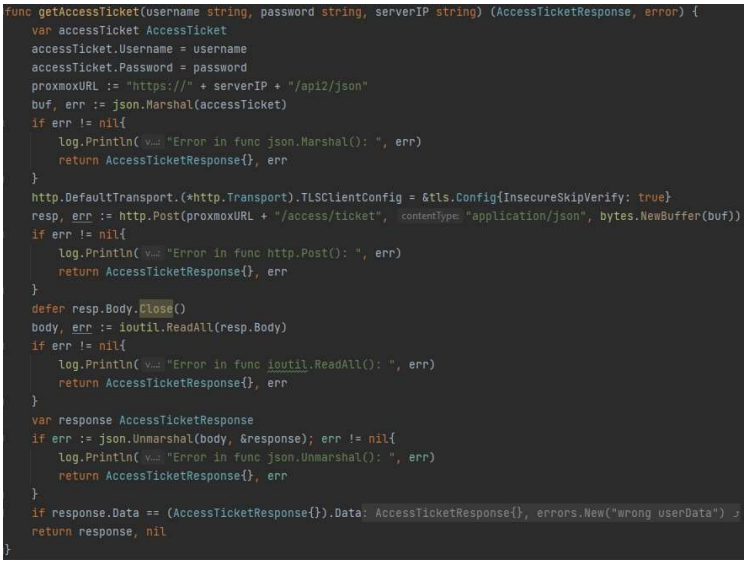

Рисунок 10 – Функция получения тикета и токена Proxmox VE

Для get-запросов написана функция, получающая на вход id пользователя и путь запроса. По id пользователя получается тикет, который добавляется в cookie. Функция возвращает ответ на запрос в виде массива байт, который конвертируется в нужную структуру уже в обработчике команд (Рисунок 11).

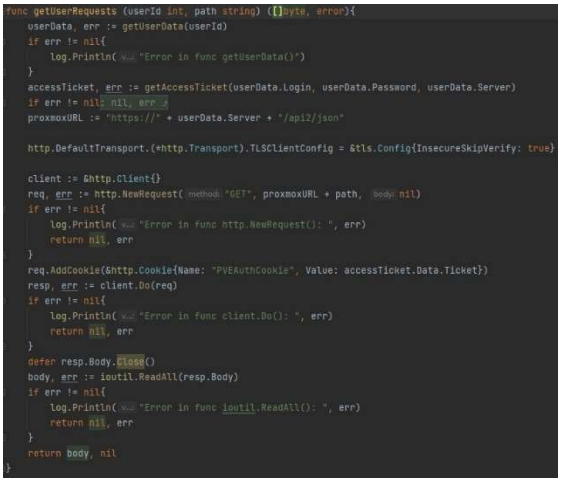

Рисунок 11 – Функция для отправки get-запросов в Proxmox VE

Для post-запросов написана функция, в аргументах которой также указываются id пользователя, путь запроса, а также может указываться аргумент типа массив байт. В последний аргумент передается уже переведенное в байтовый формат структура, для передачи в виде JSON приложения. Также в запрос добавляется заголовок с токеном. Ответ на запрос передается из функции в байтовом формате (Рисунок 12).

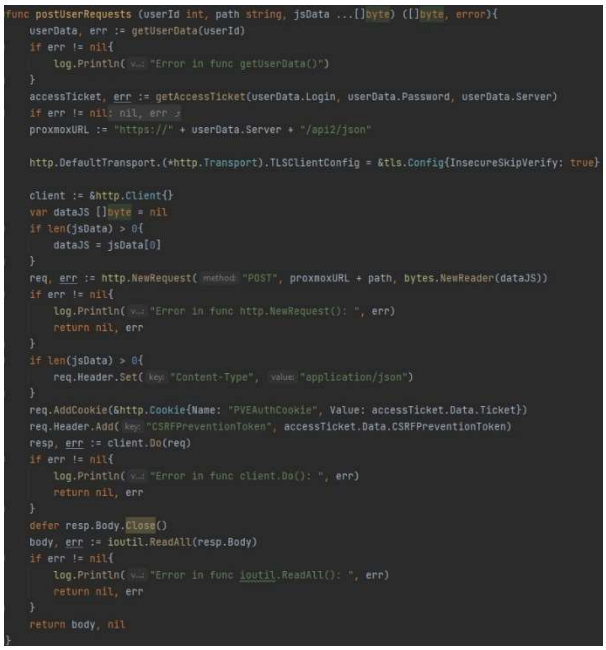

Рисунок 12 – Функция для отправки post-запросов в Proxmox VE

Тестирование

Основное тестирование проводилось параллельно этапу разработки, чтобы искать недочеты в коде сразу после написания части кода.

Финальное тестирование заключалось в использовании пользовательского меню в Telegram-боте: была введена команда “/start”, после введены данные пользователя Proxmox VE, просмотрены все существующие меню, проверено перемещение в предыдущее меню, через кнопку “Назад” (Рисунок 13).

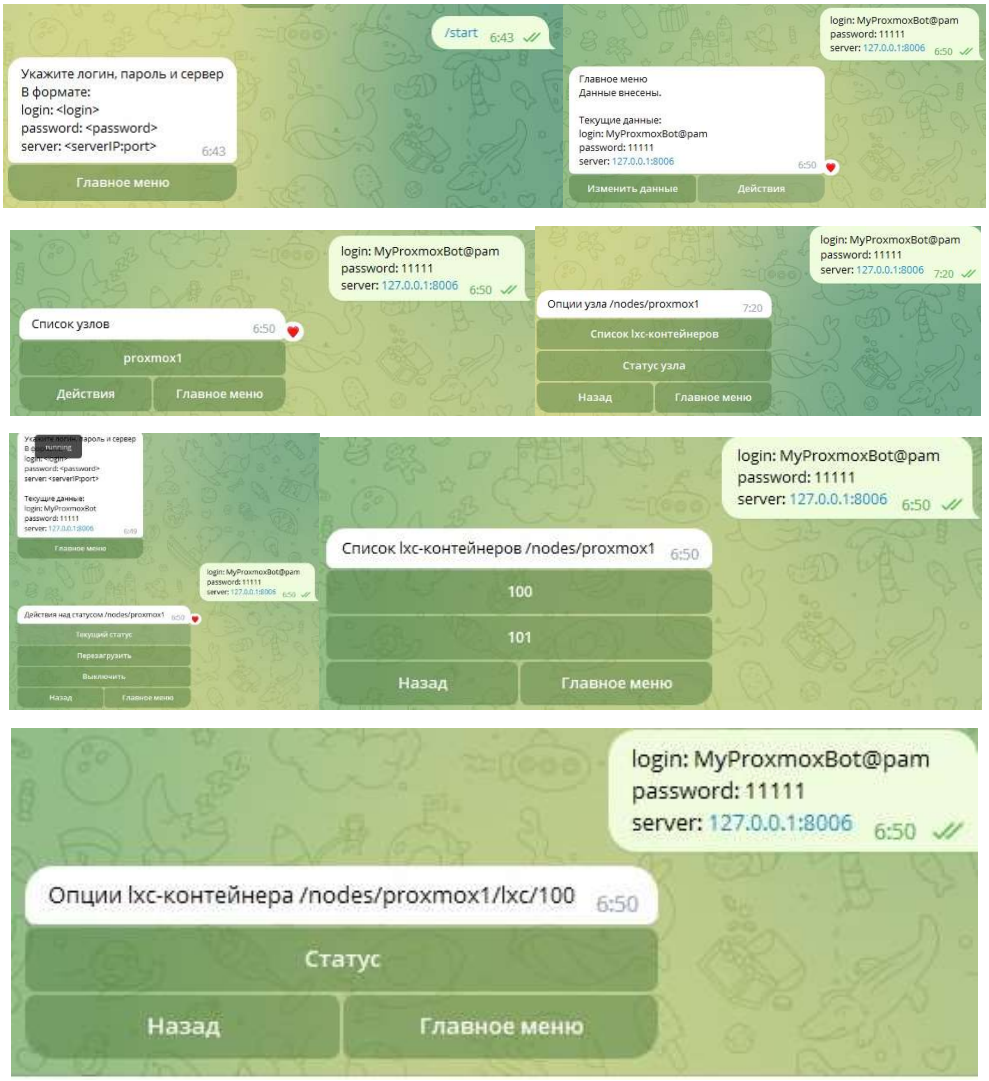

Рисунок 13 – Тестирования переходов между меню

Выполненные действия над контейнерами были проверены в логах Proxmox VE (Рисунок 14). Произведена попытка ввести пользовательские данные в неверном формате (Рисунок 15).

Node	User name	Description	Status
proxmox1	MyProxmoxBot@pam	CT 101 - Start	OK
proxmox1	MyProxmoxBot@pam	CT 100 - Shutdown	OK
proxmox1	MyProxmoxBot@pam	CT 100 - Reboot	OK
proxmox1	MyProxmoxBot@pam	CT 100 - Start	OK

Рисунок 14 – Логи на Proxmox VE

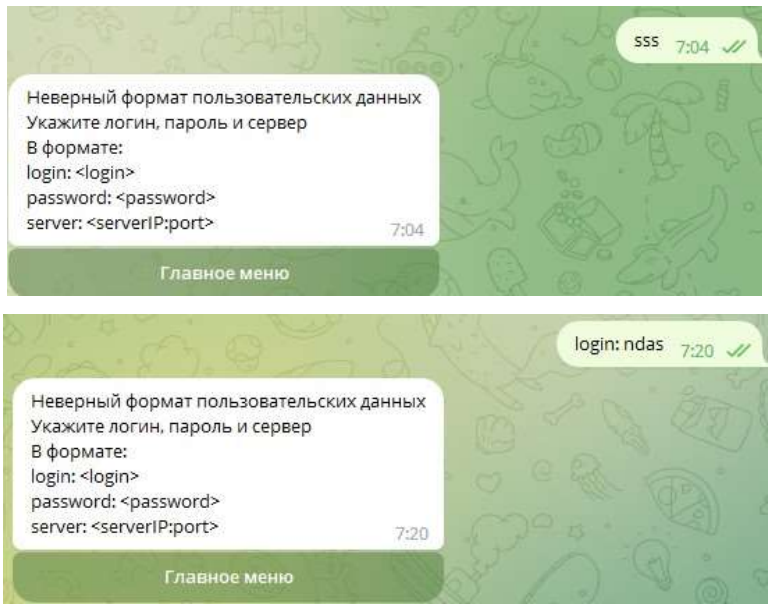

Рисунок 15 – Проверка входных данных

Выводы и заключение

В ходе практики были изучены документации для Telegram Bot API и Proxmox API, разработана часть пользовательского меню для Telegram-бота, реализовано частичное управление платформой Proxmox API. Серверная часть Telegram-бота легко масштабируется: чтобы полностью реализовать управление Proxmox VE через Telegram-бот достаточно добавить необработанные шаблоны путей в функцию определения шаблона и определить структуры ответа для них. Данная практика укрепила знания необходимые для построения клиент-серверного приложения, дала опыт использования Telegram API и Proxmox API.

Список используемых источников

1. Документация Telegram Bot API [Электронный ресурс] – Режим доступа: https://core.telegram.org/bots/ Дата доступа: 7.02.2023
2. Документация Proxmox VE API [Электронный ресурс] – Режим доступа: https://pve.proxmox.com/wiki/Proxmox_VE_API Дата доступа 16.02.2023