МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,  
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ  
   
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)  
Кафедра информационных управляющих систем

**Лабораторная работа № 02**

по дисциплине

**«ПРОГРАММИРОВАНИЕ КРИТИЧЕСКИХ СЕРВИСОВ»**

Студент гр. ИБ-03з \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Салихов К.Р.

(подпись)

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Параничев А.В.

(оценка и подпись)

Санкт-Петербург

2021 год

Лабораторная работа № 02

ЗАДАЧА.

Создание критического сервиса. Программная реализация (вариант 18).

РЕШЕНИЕ.

Код программы доступен на github:

github.com/CamilleSalikhov/telegram\_bot;

Также приложение запущено на Heroku, чтобы его запустить, перейдите по ссылке

https://telegram-botchat.herokuapp.com/

(позволяет пользоваться приложением, не запуская его у себя на машине)

Бот в телеграмме: @study\_prog\_bot

Токен для работы в боте от имени администратора (токен администратора) :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoibmFtZSIsInBhc3N3b3JkIjoicGFzc3dvcmQiLCJpYXQiOjE2NTM0NDI4MzR9.5OOOo14OELe1tQxx1K9K0V9aHj288ofb1rN2d9kLuDA

Скрипты для запуска доступны в package.json.

Запустить приложение можно командой node server.js

Для решения задачи, поставленной в предыдущей лабораторной работе, выполнено следующее:

* Добавлен модуль с основной логикой server.js
* Добавлен модель работы с базой данных database.js
* Добавлен модуль с классами classes.js
* Добавлен .env файл

Код разработанных программных модулей представлен ниже:

server.js

const {**Builder**, Browser, **By**, Key, until} = **require**('selenium-webdriver');

const **TelegramApi** = **require**('node-telegram-bot-api');

const { **MessageHandler**, **HandlerFactory**} = **require**('./classes');

**require**('dotenv').**config**();

const {**connectDB**} = **require**('./database');

const **express** = **require**('express');

const { **Options** } = **require**('selenium-webdriver/chrome');

*//для heroku*

const app = **express**();

const PORT = process.env.PORT || 5000;

app.**listen**(PORT,"0.0.0.0", console.**log**('running'));

*//mongo db соединение*

**connectDB**();

*//инициализация node-telegram-bot-api*

const token = '5246682851:AAFkeEKZB83VqHEfk0FTGuQjtsHrD75BI6c';

global.bot = new **TelegramApi**(token, {polling:true});

*//токен админа приложения*

global.adminToken = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoibmFtZSIsInBhc3N3b3JkIjoicGFzc3dvcmQiLCJpYXQiOjE2NTM0NDI4MzR9.5OOOo14OELe1tQxx1K9K0V9aHj288ofb1rN2d9kLuDA';

*//вспомогательный объект для условия IgnUser*

let ignoredChat = {

  prevMessage: '  ',

  id: '  ',

  date: 'first'

}

global.allUsers = [

  {

  name: 'admin',

  password: 'admin',

  token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoibmFtZSIsInBhc3N3b3JkIjoicGFzc3dvcmQiLCJpYXQiOjE2NTM0NDI4MzR9.5OOOo14OELe1tQxx1K9K0V9aHj288ofb1rN2d9kLuDA'

}

];

bot.**setMyCommands**([

    {command: '/login', description: 'Войти'},

    {command: '/signup', description: 'Зарегистрироваться '},

    {command: '/start', description: 'Старт'}

]);

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

*//локальные переменные*

  global.text = msg.text;

  global.substringg = text.**split**('\_')

  global.chatId = msg.chat.id;

  console.**log**(`${new **Date**()} новая дата` )

*//времени с последнего сообщения*

  if(ignoredChat.date == 'first') {

    console.**log**('Первое сообщение!');

    ignoredChat.date = new **Date**();

  } else {

    let sicnceLastMessage = new **Date**() - ignoredChat.date; *// in ms*

    sicnceLastMessage /= 1000;

    let seconds = Math.**round**(sicnceLastMessage);

    console.**log**(`С отправки последнего сообщения прошло ${seconds} секунд`)

  }

*//проверка игнюзер*

  if (ignoredChat.prevMessage == '?' && text == '?' || ignoredChat.id ==chatId  ) {

    console.**log**('начинаем игнор!');

    ignoredChat.id = chatId;

    console.**log**(ignoredChat);

    await bot.**sendMessage**(chatId, 'Ваш статус = IgnUser! Обратитесь к администратору! (для продолжения работы с ботом перезагрузите приложение или зайдите с другой учетной записи telegram)')

    return

  } else {

    ignoredChat.prevMessage = text;

    console.**log**('не игнорируем!');

    console.**log**(ignoredChat)

  }

*//if (substring[0] == '/login')*

*//основная часть*

    let messageHandler = new **MessageHandler**();

    let handlerObject = new **HandlerFactory**(substringg[0]).**handlerProducer**();

    console.**log**(` получили объект` );

    messageHandler.**setHandler**(handlerObject);

    console.**log**(messageHandler.handler)

    messageHandler.**handleMessage**();

})

database.js

const **mongoose** = **require**('mongoose');

const **connectDB** = async () => {

  try {

    const conn = await **mongoose**.**connect**('mongodb+srv://admin:admin@cscluster.jtefw.mongodb.net/?retryWrites=true&w=majority', {

      useNewUrlParser: true,

      useUnifiedTopology: true,

    })

    console.**log**('mongo db connected')

  } catch(err) {

    console.**log**(err);

  }

}

const orderSchema = new **mongoose**.**Schema**({

**name**: **String**,

**password**: **String**,

**token**: **String**,

**order**: **String**,

**date**: **String**,

**id**: **String**,

**phoneNumber**: **String**

});

const **Order** = **mongoose**.**model**('Order', orderSchema);

*// const silence = new Order({ name: 'admin' });*

*// silence.save()*

module.exports.**connectDB** = **connectDB**;

module.exports.**Order** = **Order**;

classes.js

const { logging } = **require**("selenium-webdriver");

const jwt = **require**('jsonwebtoken');

var **crypto** = **require**("crypto");

const { **Order**} = **require**('./database');

class **User** {

    constructor(text, jwt) {

        this.name = text.**split**('\_')[1];

        this.password = text.**split**('\_')[2];

        this.phoneNumber = text.**split**('\_')[3];

        this.jwt = jwt;

        this.text =text;

    }

**signUp**() {

        let payload = {

            name:this.name,

            password:this.password,

            phoneNumber: this.phoneNumber

        }

        const accessToken = this.jwt.**sign**(payload, process.env.ACCESS\_TOKEN\_SECRET);

        payload.token = accessToken;

        return payload

    }

        }

*//--------------------------------------------------------------------------------------------------------*

*// стратегия паттерн*

  class **MessageHandler** {

    constructor() {

        this.handler = {}

    }

**setHandler**(handlerObject) {

        this.handler = handlerObject;

    }

**handleMessage**() {

        console.**log**(this.handler)

        console.**log**('хендл месседж')

          return this.handler.**handleMessage**()

    }

  }

*//--------------------------------------------------------------------------------------------------------*

*// абстрактная фабрика*

  class **HandlerFactory** {

    constructor(command) {

        this.handlerType = command;

    }

**handlerProducer**() {

        let handlerObject = {};

      switch (this.handlerType) {

        case '/login': {

            handlerObject = new **LoginHandler**();

        break

        }

        case '/signup': {

            handlerObject = new **SignUpHandler**();

        break;

         }

        case  '/newaccount': {

            handlerObject = new **NewAccountHandler**();

            console.**log**('нью аккаунт')

        break;

         }

        case '/start': {

            console.**log**('ствртхендлер')

            handlerObject = new **StartHandler**();

        break;

        }

        case '/token': {

            handlerObject = new **TokenHandler**();

            break;

        }

        case '/neworder': {

            handlerObject = new **NewOrderHandler**();

        break;

         }

        case '/getorders': {

            handlerObject = new **GetOrdersHandler**();

        break;

         }

        case '/deleteorder': {

            handlerObject = new **DeleteOrderHandler**();

        break;

         }

        case '/loginadmin': {

            handlerObject = new **LoginAdminHandler**();

        break;

        }

        default : {

            console.**log**('в дефолте')

            handlerObject = new **DefaultHandler**();

        break;

        }

    }

    console.**log**(`${handlerObject} после свитча`)

    return handlerObject

    }

  }

*//-----------------------------------------------------------------------------------*

*//классы хендлеров*

  class **StartHandler** {

    constructor() {

    }

   async **handleMessage**() {

   await bot.**sendMessage**( chatId, "Здравствуйте, гость, используйте команду /login для входа , команду /signup для регистрации. Для входа в систему от имени администратора использвуйте команду \n  /loginadmin\_<token>, где <token> - это токен администратора")

        return

    }

  }

  class **DeleteOrderHandler** {

    constructor() {

    }

   async **handleMessage**() {

*// /deleteorder\_id\_token*

     const token = text.**substring**(30);

     const deleteId = substringg[1];

     console.**log**(`${token} - это токен`)

     console.**log**(deleteId)

     if (token == adminToken) {

       try {

          await **Order**.**deleteOne**({

          id: deleteId

         })

         await bot.**sendMessage**(chatId, 'Удалено!')

       } catch(err) {

         console.**log**(err)

         await bot.**sendMessage**(chatId, 'Ошибка! Возможно, ошибка формата запроса!')

       }

     } else {

       await bot.**sendMessage**(chatId, 'Ошибка доступа, удаление доступно только администратору!')

     }

return

    }

  }

  class **DefaultHandler** {

    constructor() {

    }

    async **handleMessage**() {

    await bot.**sendMessage**(chatId, "Здравствуйте, гость, используйте команду /login для входа , команду /signup для регистрации. Для входа в систему от имени администратора использвуйте команду \n /loginadmin\_<token>, где <token> - это токен администратора" );

        return

    }

  }

  class **NewOrderHandler** {

    constructor() {

    }

    async **handleMessage**() {

*///neworder\_N\_<token>*

        let payload = {};

        console.**log**(substringg);

        let orderOption = substringg[1];

        let token = text.**substring**(12);

        console.**log**(token);

        let userExists = false;

        for (let i = 0 ; i<= allUsers.length -1 ;i++) {

          if (allUsers[i] && allUsers[i].token === token) {

*//Делаем заказ, обращаемся к базе данных*

            payload = allUsers[i];

            console.**log**('нашли токен!');

            userExists = true;

            payload.order = orderOption;

         }

       }  console.**log**(payload)

       if(userExists) {

        let today = new **Date**().**toISOString**().**slice**(0, 10);

        payload.date = today;

        payload.id = **crypto**.**randomBytes**(8).**toString**('hex');

        console.**log**(payload);

        const createdOrder = new **Order**(payload);

        createdOrder.**save**()

        await bot.**sendMessage**(chatId, 'Запись создана!' );

       } else {

         await bot.**sendMessage**(chatId, 'Ошибка! Неправильный формат запроса, или такого токена не существует!' );

        }

    }

  }

  class **TokenHandler** {

    constructor() {

    }

    async **handleMessage**() {

        let token = text.**substring**(7);

        console.**log**(token);

        let accountExist = jwt.**verify**(token, process.env.ACCESS\_TOKEN\_SECRET, (err, user) => {

          console.**log**(user)

          if (err) { return false;} else { return true }

      });

        if (accountExist == false) {

        await bot.**sendMessage**(chatId, 'Такого аккаунта не существует!')

        } else {

        await bot.**sendMessage**(chatId, 'Добро пожаловать!');

        let loggedUser = '';

        allUsers.**forEach**( element => {

            if (element.token === token) loggedUser = element;

                });

        await bot.**sendMessage**(chatId, ' Напоминаем, что наш цветочный магазин предоставляет три варианта заказа: 1 - букет роз, 2 - букет лилий, 3 - букет тюльпанов. Вы можете Создать новый заказ с помощью команды /neworder\_N\_<token> , где N - Это число от 1 до 3, обозначающее вариант заказа. Запросить список ваших заказов можно с помощью /getorders\_<token>.');

                }

        return

    }

  }

  class **LoginHandler** {

    constructor() {

    }

    async **handleMessage**() {

          await bot.**sendMessage**(chatId, 'Для входа отправьте свой токен в формате \n /token\_<вашТокен>, где <вашТокен> это токен, который вы получили при регистрации и записали.');

        return

    }

  }

  class **GetOrdersHandler** {

    constructor() {

    }

    async **handleMessage**() {

*///getorders\_<token>*

        let token = text.**substring**(11);

        let adminStatus = false;

        let userExists = false;

        let payload = {};

        let ordersString = '';

          for (let i = 0 ; i<= allUsers.length -1 ;i++) {

            if (allUsers[i] && allUsers[i].token === token) {

              payload = allUsers[i];

              console.**log**('нашли токен!');

              userExists = true;

           }

         }

         if (token == adminToken ) {

           adminStatus = true;

         }

         if (userExists) {

           if(adminStatus) {

             const orders = await **Order**.**find**();

             for (let i =0; i <= orders.length -1; i++) {

               console.**log**(orders[i]);

               let currentOrder = `Пользователь "${orders[i].name}" ` + `заказал вариант "${orders[i].order}", ` + `дата: "${orders[i].date}", id "${orders[i].id}", номер телефона: "${orders[i].phoneNumber}";\n`;

        ordersString = ordersString + currentOrder;

             }

             await bot.**sendMessage**(chatId, ordersString)

           } else {

             const orders = await **Order**.**find**({

               token: payload.token

             })

             for (let i =0; i <= orders.length -1; i++) {

              console.**log**(orders[i]);

              let currentOrder = `Пользователь "${orders[i].name}" ` + `заказал вариант "${orders[i].order}", ` + `дата: "${orders[i].date}"; \n  `;

              ordersString = ordersString + currentOrder;

            }

            await bot.**sendMessage**(chatId, ordersString)

           }

         } else {

           await bot.**sendMessage**(chatId, 'Вы не зарегестрированы! Или ошибка формата запроса!')

         }

        return

    }

  }

  class **SignUpHandler** {

    constructor() {

    }

    async **handleMessage**() {

    await bot.**sendMessage**( chatId, "Гость, придумайте и введите свой логин и пароль в формате /newaccount\_name\_password\_phoneNumber");

         return

    }

  }

  class **LoginAdminHandler** {

    constructor() {

    }

    async **handleMessage**() {

        if(text.**substring**(12) == adminToken) {

          bot.**sendMessage**(chatId, 'Запросить список всех заказов можно с помощью запроса \n /getorders\_<token>, где <token> - это токен администратора ; Удалить заказ по id можно с помощью запроса \n /deleteorder\_id\_<token>, где <token> - это токен администратора')

        } else {

          await bot.**sendMessage**(chatId, 'Ошибка! Неверный формат запроса или токен!')

        }

         return

    }

  }

  class **NewAccountHandler** {

    constructor() {

    }

    async **handleMessage**() {

        let newUser = new **User**(text, jwt);

        for (let i = 0 ; i<= allUsers.length -1 ;i++) {

          if (allUsers[i] && allUsers[i].name === newUser.name) {

           return await  bot.**sendMessage**(chatId, 'Имя существует, придумайте другое!' );

        }

       }

        let signedUser = newUser.**signUp**(jwt);

        allUsers.**push**(signedUser);

        console.**log**(allUsers)

        await  bot.**sendMessage**(chatId, `Успешная регистрация,${signedUser.name}! Ваш токен "${signedUser.token}", запишите его и не потеряйте! ` );

        await  bot.**sendMessage**( chatId, "Здравствуйте, используйте команду /login для входа , команду /signup для регистрации");

         return

    }

  }

  module.exports.**MessageHandler** = **MessageHandler**;

  module.exports.**HandlerFactory** = **HandlerFactory**;

.env файл

ACCESS\_TOKEN\_SECRET = 3870196ecf0fae9c1f852b417c191d792352b8cafaf5cef1d240df4c1c69c8fd8fdabd7dc9ca1ba4d49de56dcd2df33676b777109198ccc0ac75ec62ee6464da

PORT = 3000