**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ПРОЕКТ

по дисциплине: «Введение в облачные вычисления»

на тему: **«**Университет**»**

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.

Принял: преподаватель-стажёр

Михалевич В.Г.

Гомель 2021

**Цель работы**: разработать программное обеспечение для возможности заказа блюда в ресторане быстрого питания.

**Задание:**

Необходимо разработать приложение для студентов и преподавателей ВУЗ’a (или иного учреждения образования).

Студент должен иметь возможность получать данные о собственной успеваемости: получить оценку по первой и второй аттестации за указанный студентом предмет, получить оценку по заданному экзамену, узнать состояние зачёта по соответствующему предмету, указанному студентом.

Поиск оценок студента осуществляется по его фамилии или номеру студенческого билета (реализовать оба варианта, поиск осуществлять по тому, что укажет пользователь). Результаты должны отображаться только по строго указанному предмету (не нужно выводить весь список сразу).

Преподаватели должны иметь возможность заполнить эти данные для студента по каждому предмету, иметь возможность редактировать и удалять оценки.

всю логику приложения и вычислительные операции реализовать на платформе *Node.JS*;

– в качестве хранилища данных использовать *MongoDB*;

– использовать фреймворк Express для организации *API* приложения;

– разработать методы для *CRUD*-операций (создания, чтения, обновления и удаления данных);

– обязательно организовать систему авторизации и распределения ролей;

– backend полученного приложения разместить на платформе *Azure*;

– интерфейс для управления приложением и получения данных с облака допускается реализовать в рамках android, *iOs*, *web*-приложения или чат-бота какого-либо мессенджера;

– подготовить подробный отчёт о проделанной работе, с указанием документированного листинга, демонстрацией работы приложения и его возможностей.

Для хранения данных используется база данных *MongoDB* в которой созданы три документа (*users*, *students*, *roles*) и настроена соответствующая валидация приходящих данных.

На рисунке 1 представлен пример данных документа *students* в базе данных *MongoDB*.

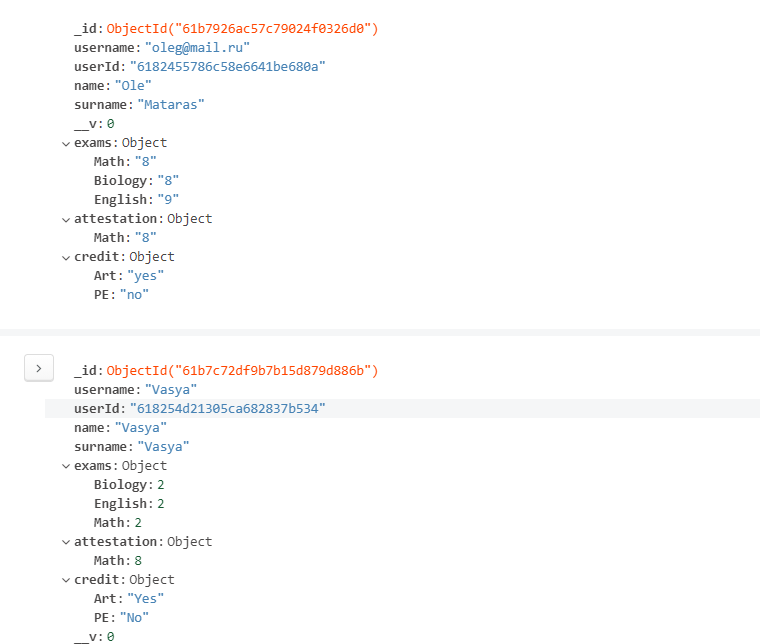


Рисунок 1 – Пример данных документа *students*

При реализации *API* с помощью *node.js* в качестве основных пакетов использованы:

*express* – для реализации сервера,

*mongoose* – для работы с базой *MongoDB*,

*jsonwebtoken* – для реализации аунтентификации на основе *JWT*,

*password-hash* – для хеширования и безопасного хранения паролей пользователей.

На рисунке 2 представлено окно клиентского приложения, где авторизованный пользователь с ролью *Teacher* можешь выбирать студентов и выставлять им оценки.

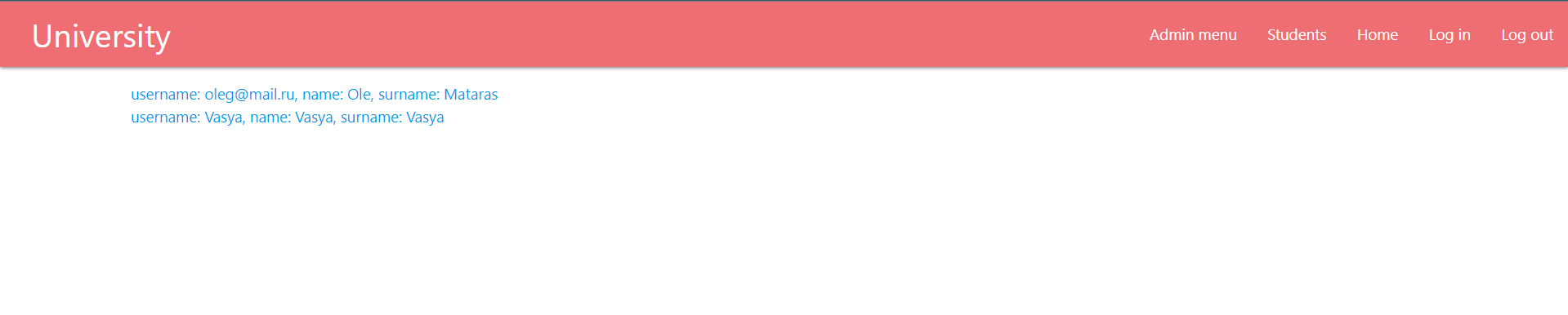


Рисунок 2 – Главное окно клиентского приложения

После выбора студента преподаватель может выставить оценки (рисунок 3).

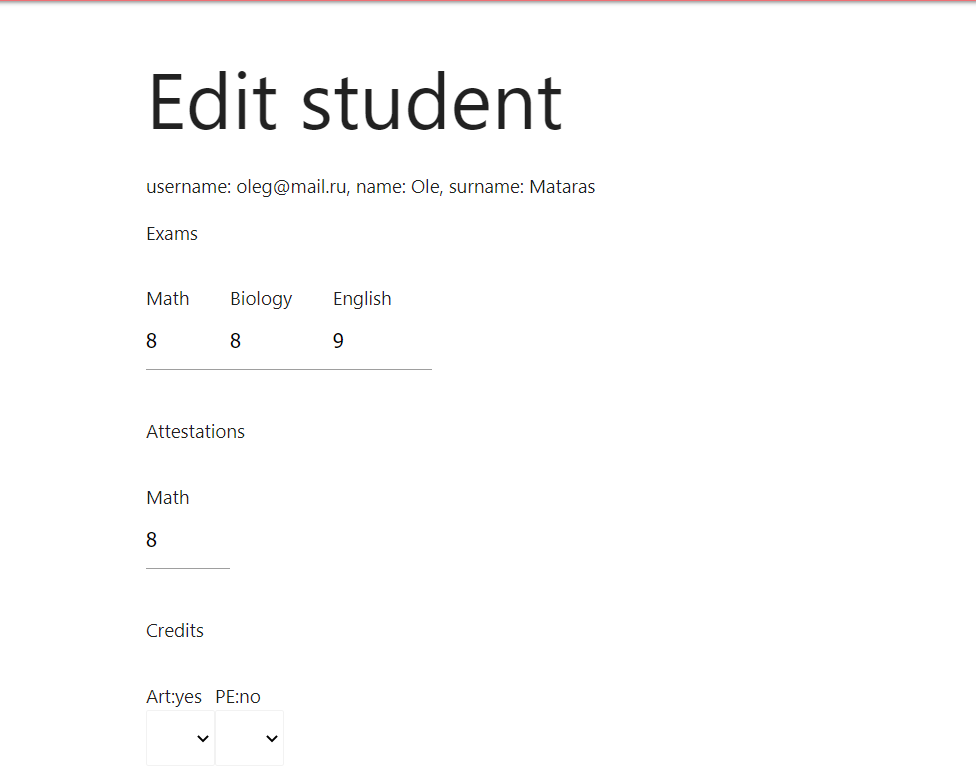


Рисунок 3 – Окно выставления оценок

Студент может просматривать свои оценки на домашней странице(*Home*), если он вошел в систему (рисунок 4).

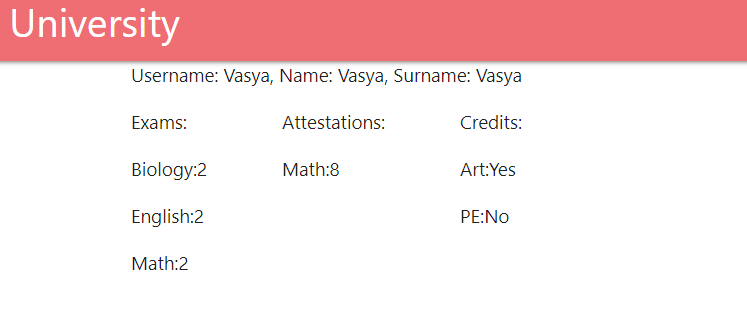


Рисунок 4 – Окно просмотра оценок студента

После окончания разработки база данных размещена на сервере *MongoDB* и доступна любому приложению, у которого есть строка подключения и пароль. Пример строки подключения представлен на рисунке 5.



Рисунок 5 – Пример строки подключения к базе данных находящейся в облаке.

Клиент созданный с помощью движка handlebars.

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения проекта изучено создание *API* с помощью *node.js*. Получены навыки отправки и обработки *HTTP*-запросов, изучено сохранение сессии пользователя, работа с аутентификацией и авторизацией пользователей.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

//Открытие сервера и подключение основных пакетов

const express = require('express')

const mongoose = require('mongoose')

const exphbs = require('express-handlebars')

const router = require('./routes/route')

const path = require('path')

const URI = 'mongodb://localhost:27017/'

const PORT = process.env.PORT || 5000

const cookieParser = require('cookie-parser')

const app = express()

const hbs= exphbs.create({

    defaultLayout:'main',

    extname:'hbs'

})

// Register `hbs.engine` with the Express app.

app.engine('hbs', hbs.engine)

app.set('view engine','hbs')

app.set('views', 'views')

app.use(cookieParser())

app.use(express.urlencoded({ extended: true }))

app.use(express.json())

app.use(express.static(path.join(\_\_dirname, 'public')))

app.use(router)

async function start(){

    try{

        await mongoose.connect(URI,{

            useNewUrlParser: true,

            useUnifiedTopology: true

        })

        app.listen(PORT, () => {

            console.log('Server has been started..')

        })

    } catch(e){

        console.log(e)

    }

}

start()

//Создание основных маршрутов

const { Router } = require("express");

const router = Router();

const controller = require("./authController");

const studController = require("./studentController");

const authMiddleware = require("../middleWare/authMiddleware");

const roleMiddleware = require("../middleWare/roleMiddleware");

const { secret } = require("./config")

const jwt = require('jsonwebtoken')

const { check } = require("express-validator");

const User = require("../models/User");

const Student = require("../models/Student");

// Registration users

router.post(

  "/registration",

  [

    check("username", "имя пользователя не может быть пустым").notEmpty(),

    check("password", "Пароль должен быть больше 4 символов").isLength({

      min: 4,

    }),

  ],

  controller.registrarion

);

router.post("/login", controller.login);

router.post("/editStudent", studController.editStudent);

router.post("/editUser", controller.editUser);

router.get("/logout", controller.logout);

router.get("/registrationView", async (req, res) => {

  res.render("registrationView", {

    title: "registration",

    isReg: true,

  });

});

router.get("/", async (req, res) => {

  let token = req.cookies.token;

  if (token) {

    let user = jwt.verify(token, secret);

    let isStudent = (user.roles == "USER") ? true: false;

    let isAdministrator = (user.roles == "ADMIN") ? true: false;

    Student.findOne({ userId: user.id }, (err, doc) => {

      if (!err) {

        res.render("index", {

          title: "index",

          user: doc,

          isStudent:isStudent,

          isIndex:true

        });

      }

    }).lean();

  }else{

    res.render("index", {

      title: "index",

      isIndex:true

    });

  }

});

router.get("/loginView", async (req, res) => {

  res.render("loginView", {

    title: "login",

    isAutorize: true,

  });

});

router.get("/adminView", roleMiddleware(["ADMIN"]), controller.getUsers);

router.get("/adminView", async (req, res) => {

  res.render("adminView", {

    title: "admin menu",

    isAdmin: true,

  });

});

router.get("/studentView", roleMiddleware(["ADMIN", "TEACHER"]), studController.getStudents);

router.get("/studentView", async (req, res) => {

  res.render("studentView", {

    title: "studentView menu",

  });

});

router.get("/studentDetail/:id", async (req, res) => {

  Student.findById(req.params.id, (err, doc) => {

    if (!err) {

      res.render("Detail", {

        title: "Detail student",

        student: doc

      });

    }

  }).lean();

});

router.get("/editUser/:id", async (req, res) => {

  User.findById(req.params.id, (err, doc) => {

    if (!err) {

      res.render("editUser", {

        title: "Edit user",

        user: doc

      });

    }

  }).lean();

});

router.get("/delete/:id", async (req, res) => {

  User.findByIdAndRemove(req.params.id, (err, doc) => {

    if (!err) {

      res.redirect("/adminView");

    }

    else {

      console.log("Error from delete " + err)

    }

  }).lean();

});

module.exports = router;