دانشکده فنی و مهندسی
کارشناسی ارشد مهندسی کامپیوتر نرم‌افزار
گروه مهندسی کامپیوتر و فناوری اطلاعات

گزارش پروژه درس سمینار

موضوع:

درگاه دانشجویی جهت اطلاع از وضعیت مالی و دروس اخذ شده

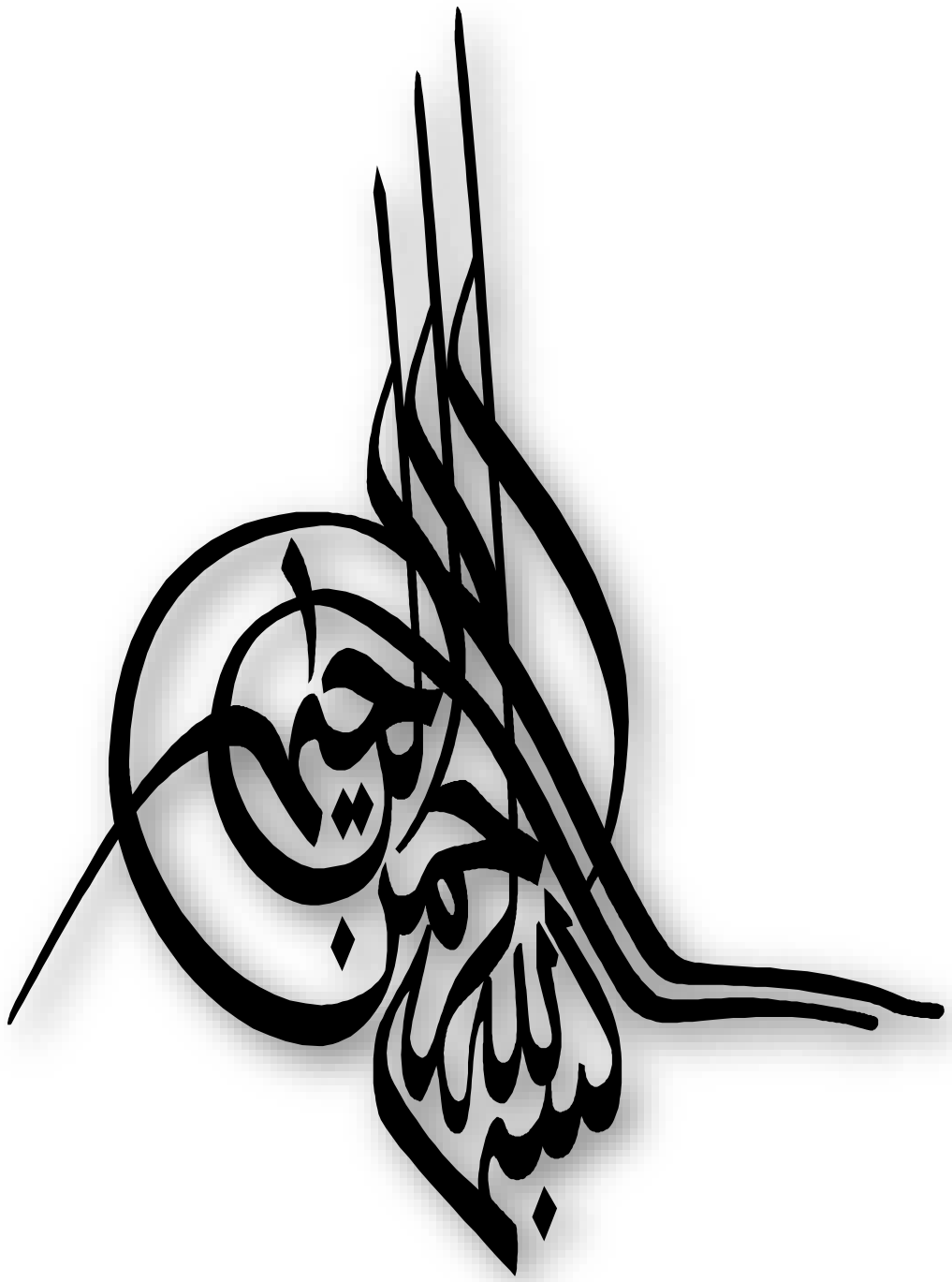
نگارش:

فهیمه ابوحمزه

استاد راهنما:

دکتر علی رضوی

خرداد ۱۴۰۰



چکیده

در راستای سهولت بخشیدن به دسترسی دانشجویان و آگاهی از کلیه پرداخت‌های خودشان و همچنین برای وجود یک مرجع واحد که سایر نهادها به راحتی بتوانند بدون کارت دانشجویی، به راحتی احراز هویت دانشجو را انجام دهند، وجود این سامانه لازم و ضروری است. هر اندازه بتوان اطلاعات را با رعایت قوانین و مقررات و حفظ امنیت در بستر اینترنت قرار دهیم، سهولت دسترسی به آن نیز به همان اندازه افزایش می‌یابد و مسائل و موارد به نگهداری کارتهای فیزیکی نیز کاهش می‌یابد. این پروژه می‌تواند دروازه‌ای به سوی جمع آوری اطلاعات کلیه دانشجویان کلیه دانشگاه‌های سراسر ایران در یک بانک اطلاعاتی باشد که به محض نیاز به این اطلاعات، هر دانشجو در هر جای این کره خاکی بتواند به آن دستیابی پیدا کند.

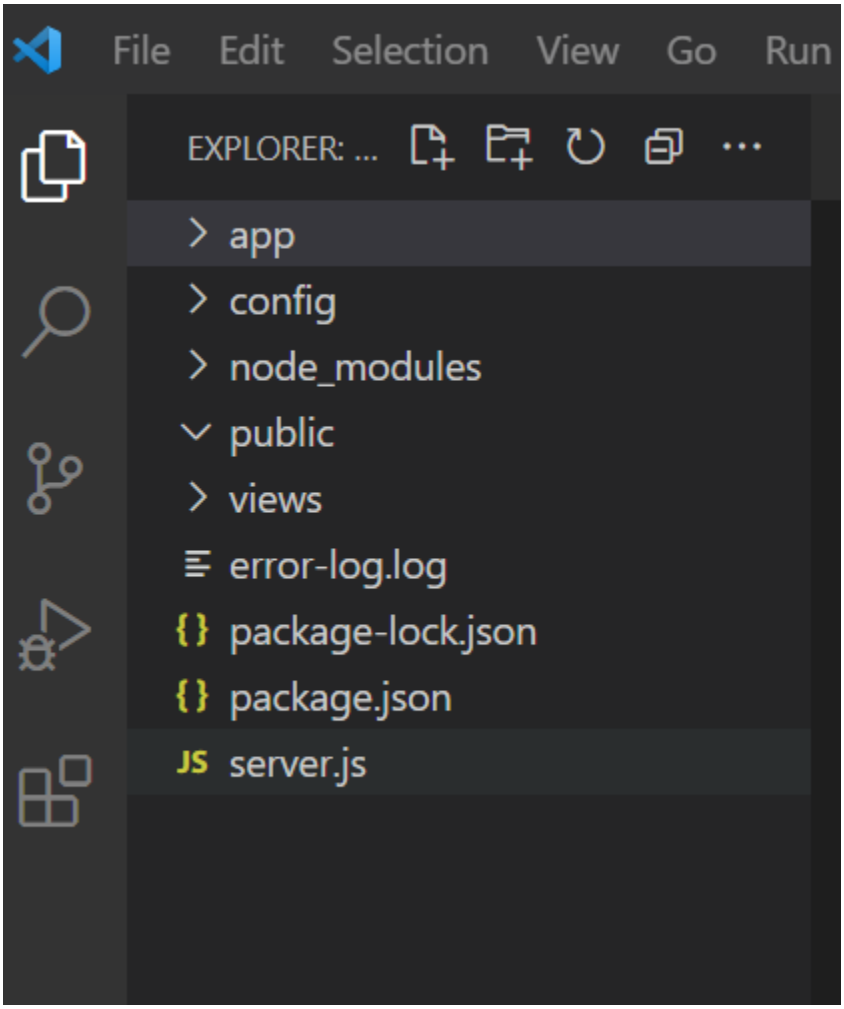
کلمات کلیدی: ، api، UI ، Controller ، Validator، Token، Route، Middelware، Model ، Config

فهرست مطالب

۱. ساختار کلی پروژه

چیدمان پوشه‌های برنامه شامل موارد زیر است.

App , config , node_modules-public-views-package.json-server.js



همانطور که در عکس بالا مشاهده می‌نمایید، کد راه‌انداز برنامه در فایل server.js قرار دارد که خود فایل index.js موجود در پوشه app را فراخوانی می‌کند.

```
const App = require ("./app");
new App()
```

۱،۱. پوشه views :

شامل کلیه صفحات رابط کاربری است. که در حال حاضر شامل فرم‌های لاگین، ثبت نام دانشجو، ایجاد رکورد پرداخت جدید و صفحه خانه است.

۱،۱. پوشه config :

شامل کد اطلاعاتی است که قرار است در سایر قسمت‌های برنامه از آن استفاده شود که در اینجا حاوی کلید خصوصی رمزگذاری برای تولید توکن است.

۱،۱. پوشه app:

مهمترین و اصلی‌ترین پوشه برنامه app است که حاوی کنترل‌ها، مدل‌های دیتابیس و آدرس‌های api است.

۱،۳،۱. پوشه models:

ساختار داکیومنت‌های دیتابیس را مشخص می‌کنیم. در این برنامه سه مدل دیتابیس تعریف شده است.

۱. Users یا همان دانشجویان،
۲. course (درس)
۳. payment

مدل payment به صورت embedded document تعریف شده است. یعنی در هر فایل json هر داکیومت اطلاعات درس و دانشجو به صورت embedded تعریف می‌شود. کد این قسمت که در فایل University.js تعریف شده است به صورت زیر است.

```
const mongoose = require ("mongoose");
const jwt = require ("jsonwebtoken");

const config = require ("config");
const schemaCourse = new mongoose.Schema({
    courseId:{type:Number , required: true},
    courseName:{type :String , required: true},
    unit :{type :Number , required:true},
    yearNo :Number ,
    amount:{type:Number , required:true}
});

const schemaPay = new mongoose.Schema({
    payId  :{type:Number, required: true},
    amount :{type :String , required: true},
    description :{type :String , required:true},
     courseId:[schemaCourse] ,
    userId:Number,
```

```
    payDate :String ,
});

schemaPay.methods.generateAuthToken= function (){
    const data ={
    _id :this._id ,
    username : this.name ,
    role :"admin"
};
return jwt.sign(data , config.get("jwtPrivateKey"));
};

const paymentModel= mongoose.model('StudentPayments' , schemaPay);
const courseModel= mongoose.model("courseModel" , schemaCourse);

module.exports={paymentModel , courseModel };
```

در این قسمت مدل payment همزمان با تعریف مدل course تعریف شده است.

و مدل دانشجو در فایل UserModel تعریف شده است.

```
const mongoose = require ("mongoose");
const jwt = require ("jsonwebtoken");
const config = require ("config");
const UserSchema = new  mongoose.Schema({
     code : String ,
     fullName : {type : String , required : true}  ,
     phone :  {type : String , unique: true , required : true },
     password : {type : String , required : true} ,
     email:{type:String , required : true} ,
     role : { type :String , required : true , default :"user"} ,
     active :{type :Boolean , default:false}

});

UserSchema.methods.generateAuthToken= function (){
          const data ={
          _id :this._id ,
          fullName : this.fullName ,
          role : this.role
     };
     return jwt.sign(data , config.get("jwtPrivateKey"));
```

```
};

const UserModel= mongoose.model("User" , UserSchema);

module.exports=UserModel;
```

۱،۳،۲. پوشه Route:

در قسمت routes شامل routeهای هر ماجول برنامه است که در سه بخش:

۱. Api.js
۲. CourseRoutes.js
۳. UserRoutes.js
۴. PaymentRoutes.js

در فایل api.js بخش ثابت هر مسیر api مشخص میشود. که

```
const router = require ("express").Router();

const PaymentRoutes = require ("./PaymentRouts");
const UserRoutes = require ("./UserRoutes");
const CourseRoutes = require ("./CourseRoutes");
router.use("/student" ,PaymentRoutes ) ;
router.use("/student" ,UserRoutes ) ;
router.use("/course" ,CourseRoutes ) ;
module.exports= router;
```

مثلا بخش ثابت پرداخت student است و معنی آن این است که هر api با توجه به نوعش شامل این قسمت هست.
قبل از این مسیر یک بخش دیگر هم هست که برای کلیه apiها هست، که در قسمت index فایل راه انداز برنامه قرار دارد.

```
app.use("/api" , api);
```

در این قسمت لازمه درست بودن آدرس داشتن قسمت api/ است.

Routeهای بخش course و payment شبیه به هم است و به صورت زیر تعریف میشود که شامل موارد زیر است:

۱. لیست دروس و پرداختها (GetList)
۲. فراخوانی یک درس یا پرداخت با شناسه یکتای آن (GetOne)
۳. ایجاد رکورد درس و پرداخت (Post)

۴. بروزرسانی رکورد درس و پرداخت (Put)

۵. حذف رکورد درس و پرداخت(Delete)

```
 const router = require ("express").Router();

const Auth = require ("../http/middleware/Auth");
const admin= require("../http/middleware/UniversityAdmin");
const controller = require("../http/controller/PaymentController");
router.get('/' ,controller.getPaymentList);
router.get('/:id' , controller.getOne);
router.post('/' , controller.create);
router.put('/:id' ,controller.update );
router.delete('/:id' , controller.delete);

module.exports= router;
```

قسمت User برنامه شامل دو route :

۱. Register

۲. Login

```
3. const express = require("express");
4. const router = express.Router();
5. const UserController = require('../http/controller/UserController')
6. const Auth = require("../http/middleware/Auth");
7.
8. router.post("/register", UserController.registerUser);
9. router.post("/login", UserController.loginUser);
10.
11.module.exports = router;
```

۱،۳،۳. کنترلرها

کد مربوط به هر route در قسمت controller تعریف شده است. با توجه به routeهای موجود ۳ تا کنترلر داریم:

۱. UserController

۲. PaymentController

۳. CourseController

```
const {courseModel} = require("../../models/University");
const _ = require("lodash");
const {validateCreateCourse ,validateUpdateCourse } = require ("../validator/Cour
seVlidator")

class CourseController {

    async getList(req , res){
        const list = await courseModel.find()
        .select("courseId  courseName unit yearNo amount" )
        .limit(20);
        res.send(list);
     }

async addone(req , res){
    console.log(req.user);
    res.send("hi")
}



async getOne(req , res){
    const id = req.params.id;
    const data = await courseModel.findById(id)
    .select("courseId  courseName unit yearNo amount " );  //   select("-
adminPassword" );
    if(!data) return res.status(404).send("not found");

    res.send(data);
}


async create(req , res){
    const {error} = validateCreateCourse(req.body);
    if(error) return res.status(400).send(error.message);
    let Course = new courseModel(_.pick (req.body ,[
        "courseId" ,
        "courseName" ,
        "unit",
        "yearNo",
        "amount"
      ])
    );

    Course= await Course.save();
```

```
    res.send(Course);
   }

   async update(req , res){
    const id = req.params.id;
    const {error} = validateUpdateCourse(req.body);
    if(error) return res.status(400).send(error.message);
    const result = await  courseModel.findByIdAndUpdate(id , {
        $set : _.pick (req.body ,[
            "courseId" ,
            "courseName" ,
            "unit",
            "yearNo",
            "amount"

      ])
        },{new :true});
        if (!result) return res.status(404).send("not found!")
        res.send(_.pick (result ,[
            "courseId" ,
            "courseName" ,
            "unit",
            "yearNo",
            "amount"

          ])
           );
        }



async delete(req , res){

}

}

module.exports= new CourseController;
```

در این قسمت یک کلاس با تمام متدهای هر route تعریف می‌شود.

۱,۳,۴. Validator:

برای متدهای create و update ابتدا دیتا صحت سنجی می‌شود. برای این کار از فایل‌های validator که برای هر route نوشته شده استفاده می‌شود. در این کدها از کتابخانه Joi استفاده شده است که نوع پارامتر ورودی را از لحاظ اینکه عدد باشد یا حرفی یا حداقل و حداکثر طول مورد نظر و اینکه ضروری هست یا خیر را بررسی می‌کند.

```
const Joi = require("joi");
const joiObjectid = require("joi-objectid");
 Joi.objectId = require("joi-objectid")(Joi);

const validateCreatePayment=(data)=>{
    const schema = Joi.object({
        payId :Joi.number().required(),
        amount:Joi.number().required(),
        description :Joi.string(),
       //userId :Joi.required(),
       //courseId :Joi.required(),
        payDate :Joi.string().length(10).required(),
    });
    return schema.validate(data);
};

const validateUpdatePayment=(data)=>{
    const schema = Joi.object({
        payId :Joi.number(),
        amount:Joi.number(),
        description :Joi.string(),
       //userId :Joi.required(),
       //courseId :Joi.required(),
        payDate :Joi.string().length(10),

    });
    return schema.validate(data);
};

module.exports={validateCreatePayment,validateUpdatePayment  };
```

در این برنامه هر زمان که کاربر به برنامه لاگین میکند یک token با استفاده از کتابخانه generateAuthToken ساخته و در local storage در قسمت header نگه‌داشته می‌شود. برای ملزم کردن برنامه به وجود ارسال توکن تولید شده از middleware auth استفاده می‌شود :

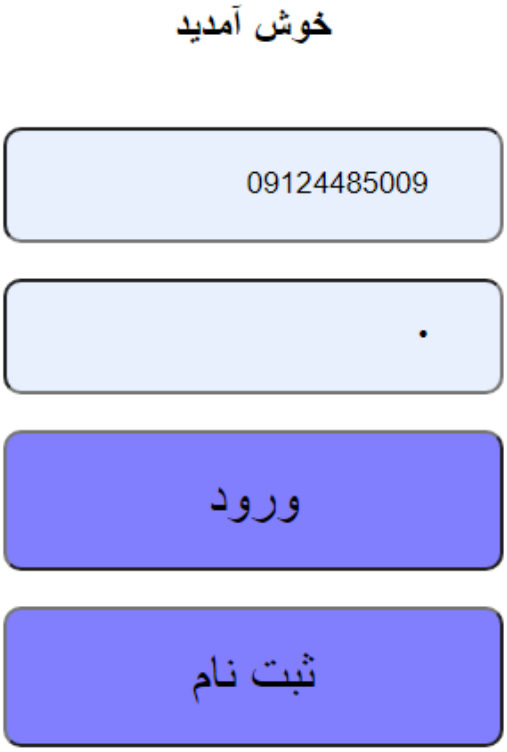
```
const jwt = require ("jsonwebtoken");
const config = require ("config");
module.exports = function(req , res , next ){
    const token = req.header("x-auth-token");
    if (!token)  return res.status(401).send ("شما اجازه دسترسی به این بخش را ندارین");

    try {
        const user = jwt.verify(token , config.get("jwtPrivateKey"));
        req.user = user;
        next();
    }
    catch(ex){
        return res.status(401).send ("شما اجازه دسترسی به این بخش را ندارین");
    }

}
```

برای اینکه بتوان الزام ارسال کد توکن را اجباری کرد از این middleware در قسمت تعریف router استفاده شده است.

```
router.get("/api/students",Auth, StudentController.StudentList);
```

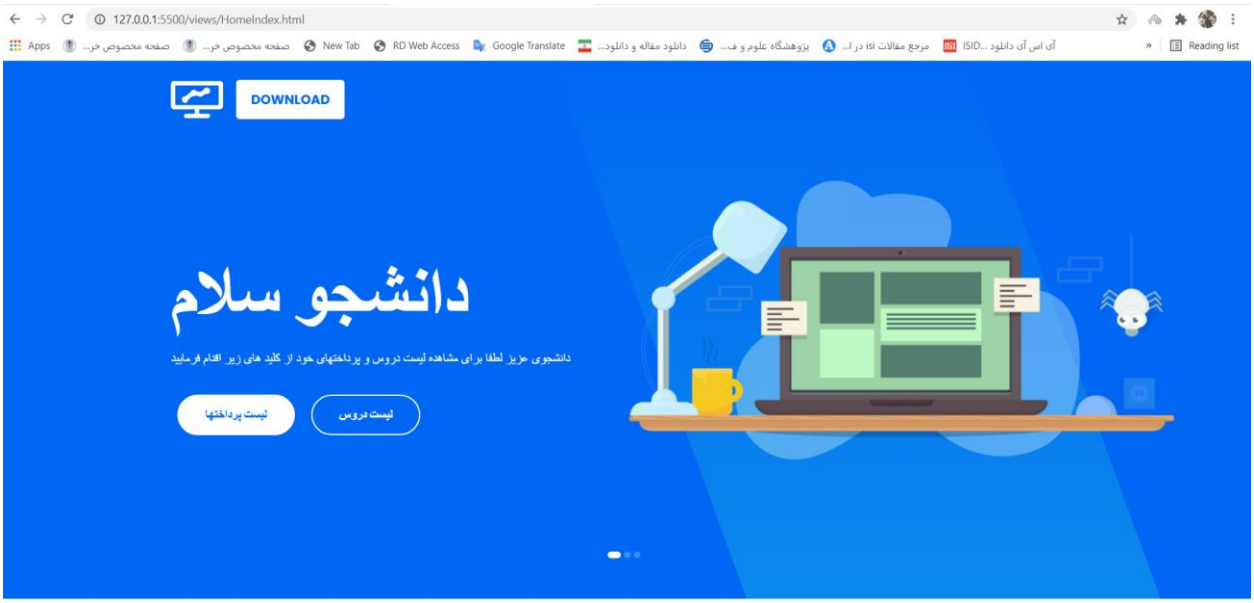
۲. رابط کاربری برنامه

برنامه حاضر دارای صفحه ورود به پنل دانشجویی است. در خود صفحه لاگین در صورتیکه دانشجو کد کاربری ندارد می‌تواند خودش با زدن کلید ایجاد کاربری، اقدام به ایجاد حساب کاربری در سایت حاضر انجام دهد.



در صورت زدن کلید ثبت نام سیستم فرم ایجاد دانشجوی جدید باز می‌شود.

در صورتیکه موبایل شخص تکراری باشد سیستم پیام مناسب مبنی بر تکراری بودن را صادر می‌کند. بعد از پیام موفقیت آمیز بودن، دوباره صفحه لاگین باز می‌شود و بعد از وارد کردن نام کاربری و رمز ورود صفحه اصلی برنامه که به صورت زیر است باز می‌شود:



با زدن کلید لیست پرداخت‌ها، لیست پرداخت‌های هر دانشجو باز می‌شود.

ایجاد	توضیحات	کاربر وارد کننده	مبلغ	تاریخ پرداخت	شماره پرداخت
	salam	undefined	123	1400/03/15	1
	salam	undefined	123	1400/03/15	1
	salam	undefined	123	1400/03/15	1
	salam	undefined	123	1400/03/15	1
	salam	undefined	123	1400/03/15	1
	adsfasdf	undefined	963258	1400/03/16	5895
	salam	undefined	123	1400/03/15	1
	salam	undefined	123	1400/03/13	1
	asdfasfasdfas	undefined	632000	1400/02/03	9966
	این پرداخت جهت خرید خانه است	undefined	8900000	1400/03/21	326500
	این پرداخت جهت خرید خانه است	undefined	6500000	1400/04/21	96000

که در این پنجره هر دانشجو با توجه به دسترسی می‌تواند اقدام به ایجاد، ویرایش و حذف کند.

با زدن کلید ایجاد پنجره زیر باز می‌شود:

پرداخت جدید

شماره پرداخت :

مبلغ:

توضیحات :

تاریخ پرداخت :

درج

با سپاس از همراهی شما