ФГБОУ ВО «КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ, ВЫПОЛНЯЕМОЕ В ПЕРИОД ПРОВЕДЕНИЯ УЧЕБНОЙ ПРАКТИКИ
Научно-исследовательской работы
(получение первичных навыков научно-исследовательской работы)

Студент Козин Александр Александрович
(фамилия, имя, отчество полностью)
Направление подготовки (специальности) 02.03.03 Математическое обеспечение и администрирование информационных систем

Место прохождения практики компьютерные классы факультета компьютерных технологий и прикладной математики ФГБОУ ВО «КубГУ»

Срок прохождения практики с 06.07.2022 г. по 19.07.2022 г.

Цель практики – Закрепление теоретических знаний, полученных при изучении предметов «Разработка пользовательского WEB интерфейса», «Компьютерные сети», «Аппаратно-программные средства WEB»; изучение студентом деятельности по анализу литературы, сбору данных и построению алгоритмов решения практических задач; проверка степени готовности будущего бакалавра к самостоятельной работе; приобретение практических навыков (опыта практической деятельности) в использовании знаний, умений и навыков по программированию; воспитание устойчивого интереса к профессии, убежденности в правильности ее выбора; овладение профессиональными навыками работы; выбор направления практической работы; сбор необходимой для выполнения данной работы информации по месту прохождения практики, а также при изучении литературных и иных источников; приобретение опыта работы в коллективе; подготовка студентов к последующему осознанному изучению профессиональных, в том числе профильных дисциплин.
Формирование компетенций, регламентируемых ФГОС ВО:

Код компетенции	Содержание компетенции (или её части)	Планируемые результаты при прохождении практики
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	
ОПК-3	Способен применять современные информационные технологии, в том числе отечественные, при создании программных продуктов и программных комплексов различного назначения	
ПК-1	Способен демонстрировать базовые знания математических и естественных наук, программирования и информационных технологий	
ПК-2	Способен проводить под научным руководством исследование на основе существующих методов в конкретной области профессиональной деятельности	

ПК-3	Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла	

1. Способностью к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, текстов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям.
Перечень вопросов (заданий, поручений) для прохождения практики

2. Проект представляет собой клон российского интернет-издания и агрегатора новостей TJ (tjournal).

Ознакомлен _____ Козин А.А.
подпись студента *расшифровка подписи (ФИО)*

Руководитель учебной практики
доцент кафедры информационных технологий
факультета компьютерных технологий
и прикладной математики, к.ф.-м.н., доцент _____ Лукащик Е.П.

Рабочий график (план) проведения практики:

№	Этапы работы (виды деятельности) при прохождении практики	Сроки	Отметка руководителя практики от университета о выполнении (подпись)
1	Инструктаж по технике безопасности, охраны труда, пожарной безопасности, а также правилами внутреннего распорядка обучающихся. Ознакомление с календарным планом, программой учебной практики, ее целями и задачами. Составление календарно-тематического плана прохождения практики.	06.07.2022 г.- 07.07.2022 г.	
2	Детальное изучение условий задачи, сбор и систематизация теоретического материала и статистической информации по исследуемой проблеме, анализ, формализация и выбор математических материалов решения задачи.	08.07.2022 г.- 10.07.2022 г.	
3	Выбор и детализация информационной модели для представления данных решаемой задачи. Составление программы, её отладка и проведение тестовых расчетов.	11.07.2022 г.- 15.07.2022 г.	
4	Оформление отчета.	16.07.2022 г.- 18.07.2022 г.	
5	Защита отчета о практике в срок до	19.07.2022 г.	

Ознакомлен _____ Козин А.А.
подпись студента *расшифровка подписи (ФИО)*

«06» июля 2022 г.

Руководитель учебной практики
доцент кафедры информационных технологий
факультета компьютерных технологий
и прикладной математики, к.ф.-м.н., доцент _____ Лукащик Е.П.

ОЦЕНОЧНЫЙ ЛИСТ
результатов прохождения учебной практики
научно-исследовательской работы
(получение первичных навыков научно-исследовательской работы)
по направлению подготовки
02.03.03 Математическое обеспечение и администрирование информационных систем

Фамилия И.О студента Козин Александр Александрович
Курс 2

№	ОБЩАЯ ОЦЕНКА (отмечается руководителем практики)	Оценка			
		5	4	3	2
1.	Уровень подготовленности студента к прохождению практики				
2.	Умение правильно определять и эффективно решать основные задачи				
3.	Степень самостоятельности при выполнении задания по практике				
4.	Оценка трудовой дисциплины				
5.	Соответствие программе практики работ, выполняемых студентом в ходе прохождении практики				

№	СФОРМИРОВАННЫЕ В РЕЗУЛЬТАТЕ учебной практики (практики по получению первичных профессиональных умений и навыков) ПРАКТИКИ КОМПЕТЕНЦИИ (отмечается руководителем практики от университета)	Оценка			
		5	4	3	2
1.	ОПК-1 Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности				
2.	ОПК-3 Способен применять современные информационные технологии, в том числе отечественные, при создании программных продуктов и программных комплексов различного назначения				
3.	ПК-1 Способен демонстрировать базовые знания математических и естественных наук, программирования и информационных технологий				
4.	ПК-2 Способен проводить под научным руководством исследование на основе существующих методов в конкретной области профессиональной деятельности				
5.	ПК-3 Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла				

Руководитель учебной практики
доцент кафедры информационных технологий
факультета компьютерных технологий
и прикладной математики, к.ф.-м.н., доцент _____ Лукащик Е.П.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный университет
Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

ОТЧЕТ О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ
научно-исследовательской работы
(получение первичных навыков научно-исследовательской работы)

период с 06.07.2022 г. по 19.07.2022 г.

Козин Александр Александрович
(Ф.И.О. студента)

студента 25/2 группы 2 курса ОФО

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Руководитель учебной практики
доцент кафедры информационных технологий
факультета компьютерных технологий
и прикладной математики, к.ф.-м.н., доцент _____ Лукащик Е.П.
ученое звание, должность *(подпись)* *(Ф.И.О)*

Оценка по итогам защиты практики: _____

«_____» _____ 2022 г.
(дата)

Краснодар 2022 г.

Стек технологий

Среда разработки JetBrains WebStorm. Она поддерживает JS, CSS & HTML.

Веб сервер Apache, связываюсь с MySQL посредством веб-приложения phpMyAdmin.

Технологии, используемые при создании сайта.

Frontend:

- NextJS / React
- TypeScript
- Redux / Redux Toolkit
- EditorJS
- React Hook Form / Yup
- Material UI
- SCSS / CSS-modules
- Nookies
- Axios

Backend:

- NestJS
- TypeScript
- MySQL
- JWT / PassportJS
- Class-validator/transformer

HTML – стандартизированный язык разметки документов для просмотра веб-страниц в браузере.

CSS – формальный язык описания внешнего вида документа, написанного с использованиям языка разметки.

JavaScript – мультипарадигменный язык программирования.

React – это JavaScript-библиотека для создания пользовательских интерфейсов.

React Hook Form – эффективная, гибкая и расширяемая форма с простой валидацией.

Yup – библиотека для валидации объектов в js.

NextJS – это фреймворк, основанный на React, который позволяет создавать веб-приложения с улучшенной производительностью и улучшенным пользовательским опытом с помощью дополнительных функций предварительного рендеринга, таких как полноценный **рендеринг на стороне сервера** (SSR) и **статическая генерация страниц** (SSG).

TypeScript – это расширенная версия JavaScript, главной целью которого является упрощение разработки крупных JS-приложений. Этот язык добавляет много новых принципов — классы, дженерики, интерфейсы, статические типы, что позволяет разработчикам использовать разные инструменты, такие как статический анализатор или рефакторинг кода.

Redux – библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения.

Redux Toolkit – это набор как специально разработанных инструментов, которые обычно используются совместно с Redux.

Nookies 🍪 – библиотека для работы с куками.

EditorJS – визуальный редактор.

Material UI – это набор компонентов React, который реализует Google Material Design.

SCSS – это “диалект” языка SASS.

SASS – это метаязык на основе CSS-кода и упрощения файлов каскадных таблиц стилей.

Axios – это библиотека позволяющая делать HTTP-запросы.

NestJS – это фреймворк для создания эффективных масштабируемых Node.js серверные приложения, созданные с использованием TypeScript и полностью поддерживающие его.

MySQL — свободная реляционная система управления базами данных.

JSON Web Token (JWT) – это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером,

подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

PassportJS – это связующее программное обеспечение для авторизации под Node.js.

Class-validator/transformer - Позволяет использовать проверку на основе декоратора и без декоратора. Внутренне использует validator.js для выполнения проверки. Позволяет преобразовать обычный объект в какой-либо экземпляр класса и наоборот.

Функционал сайта

Мною был выбран функционал сайта, который я хочу реализовать в данной практике. Функционал описан ниже, и схема создана с помощью сайта MindMeister и представлен ниже:

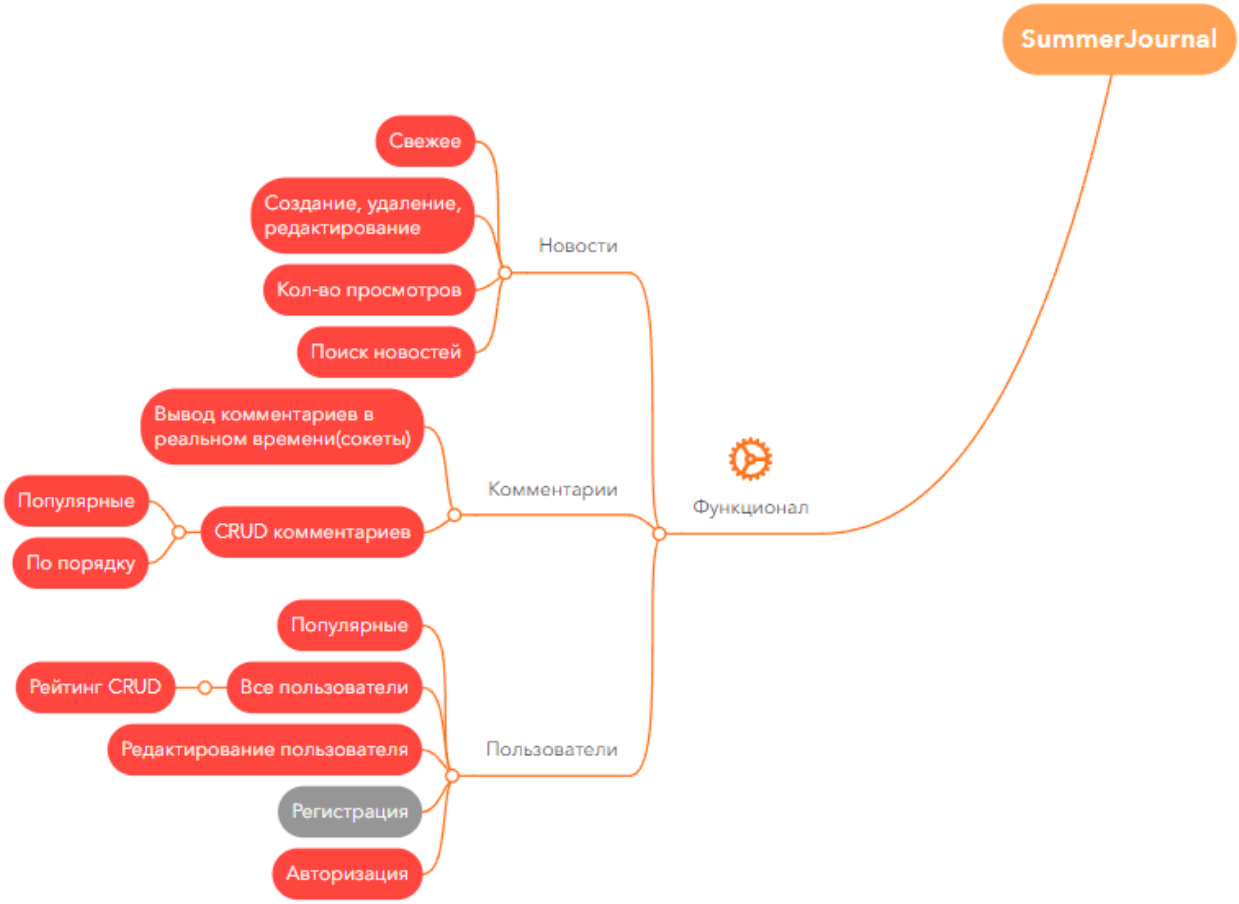
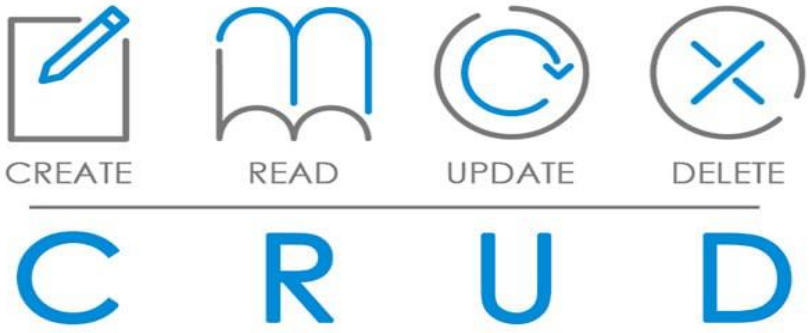


Рисунок 1 – Графическое представление функционала сайта

CRUD – акроним, обозначающий 4 базовые функции, используемые при работе с базами данных: создание, чтение, модификация, удаление.



СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Синица, С. Г. Веб-программирование и веб-сервисы: учебное пособие / С. Г. Синица; М-во образования и науки Рос. Федерации, Кубанский гос. Ун-т, 2013. - 158 с. – ISBN XXX-X-XXXX-XXXX-X

2 Стандарты оформления исходного кода программ и современные интегрированные среды разработки программного обеспечения: учеб.-метод.пособие. Ю.В. Кольцов, А.В.Уварова, С.Г.Синица [и др.]; М-во образования и науки Рос. Федерации, Кубанский гос. ун-т. - Краснодар: [Кубанский государственный университет], 2017. – ISBN XXX-X-XXXX-XXXX-X

3 Лукащик, Е. П. Основы администрирования информационных сетей: учебно-методическое пособие / Е. П. Лукащик, О. И. Ефремова; М-во образования и науки Рос. Федерации, Кубанский гос. ун-т. - Краснодар: [Кубанский государственный университет], 2014. - 45 с. – ISBN XXX-X-XXXX-XXXX-X

4 Бессарабов Н.В. Базы данных: модели, языки, структуры и семантика. М.: «ИНТУИТ», 2013. 523 с – ISBN XXX-X-XXXX-XXXX-X

5 Помощь по Microsoft Office: Microsoft Office Word. [Электронный ресурс] – URL: https://support.microsoft .com/ru-ru/word. (25.05.2022)

6 Material-ui documentation: https://v4.mui.com/ru/getting-started/installation/

7 NestJS documentation: https://docs.nestjs.com/

8 NextJs documentation: https://nextjs.org/docs

ПРИЛОЖЕНИЕ

Код программы(backend)

package.json

```
{
  "name": "summer-journal-backend",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "prebuild": "rimraf dist",
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register
node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^8.0.0",
    "@nestjs/core": "^8.0.0",
    "@nestjs/jwt": "^8.0.0",
    "@nestjs/mapped-types": "*",
    "@nestjs/passport": "^8.2.1",
    "@nestjs/platform-express": "^8.0.0",
    "@nestjs/typeorm": "^8.0.3",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.13.2",
    "mysql2": "^2.3.3",
    "passport": "^0.5.2",
    "passport-jwt": "^4.0.0",
    "passport-local": "^1.0.0",
    "reflect-metadata": "^0.1.13",
    "rimraf": "^3.0.2",
    "rxjs": "^7.2.0",
    "typeorm": "0.2"
  },
```

```
  "devDependencies": {
    "@nestjs/cli": "^8.0.0",
    "@nestjs/schematics": "^8.0.0",
    "@nestjs/testing": "^8.0.0",
    "@types/express": "^4.17.13",
    "@types/jest": "27.4.1",
    "@types/node": "^16.0.0",
    "@types/passport-jwt": "^3.0.6",
    "@types/passport-local": "^1.0.34",
    "@types/supertest": "^2.0.11",
    "@typescript-eslint/eslint-plugin": "^5.0.0",
    "@typescript-eslint/parser": "^5.0.0",
    "eslint": "^8.0.1",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-prettier": "^4.0.0",
    "jest": "^27.2.5",
    "prettier": "^2.3.2",
    "source-map-support": "^0.5.20",
    "supertest": "^6.1.3",
    "ts-jest": "^27.0.3",
    "ts-loader": "^9.2.3",
    "ts-node": "^10.0.0",
    "tsconfig-paths": "^3.10.1",
    "typescript": "^4.3.5"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
      "**/*.(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}
```

main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors({
    origin: [/^(.*)/],
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    preflightContinue: false,
    optionsSuccessStatus: 200,
    credentials: true,
    allowedHeaders:
      'Origin,X-Requested-With,Content-Type,Accept,Authorization,authorization,X-Forwarded-
for',
  });

  app.useGlobalPipes(new ValidationPipe());
  await app.listen(3001);
}

bootstrap();
```

app.service.ts

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}
```

app.controller.ts

```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';
@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();}
}
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { UserModule } from './user/user.module';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UserEntity } from './user/entities/user.entity';
import { PostModule } from './post/post.module';
import { PostEntity } from './post/entities/post.entity';
import { CommentModule } from './comment/comment.module';
import { CommentEntity } from './comment/entities/comment.entity';
import { AuthModule } from './auth/auth.module';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'mysql',
      host: 'localhost',
      port: 3306,
      username: 'root',
      password: '',
      database: 'sjournal',
      entities: [UserEntity, PostEntity, CommentEntity],
      synchronize: true,
    }),
    UserModule,
    PostModule,
    CommentModule,
    AuthModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

USER

user.service.ts

```
import { Injectable } from '@nestjs/common';
import { CreateUserDto } from './dto/create-user.dto';
import { UpdateUserDto } from './dto/update-user.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { UserEntity } from './entities/user.entity';
import { Repository } from 'typeorm';
import { LoginUserDto } from './dto/login-user.dto';
import { SearchUserDto } from './dto/search-user.dto';
import { CommentEntity } from '../comment/entities/comment.entity';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(UserEntity)
    private repository: Repository<UserEntity>,
  ) {}

  create(dto: CreateUserDto) {
    return this.repository.save(dto);
  }

  async findAll() {
    const arr = await this.repository
      .createQueryBuilder('u')
      .leftJoinAndMapMany(
        'u.comments',
        CommentEntity,
        'comment',
        'comment.userId = u.id',
      )
      .loadRelationCountAndMap('u.commentsCount', 'u.comments', 'comments')
      .getMany();

    return arr.map((obj) => {
      delete obj.comments;
      return obj;
    });
  }

  findById(id: number) {
    return this.repository.findOne(id);
  }

  findByCond(cond: LoginUserDto) {
    return this.repository.findOne(cond);
```

```
  }

  update(id: number, dto: UpdateUserDto) {
    return this.repository.update(id, dto);
  }

  async search(dto: SearchUserDto) {
    const qb = this.repository.createQueryBuilder('u');

    qb.limit(dto.limit || 0);
    qb.take(dto.take || 10);

    if (dto.fullName) {
      qb.andWhere(`u.fullName LIKE :fullName`);
    }

    if (dto.email) {
      qb.andWhere(`u.email LIKE :email`);
    }

    qb.setParameters({
      email: `%${dto.email}%`,
      fullName: `%${dto.fullName}%`,
    });

    const [items, total] = await qb.getManyAndCount();

    return { items, total };
  }
}
```

user.module.ts

```
import { Module } from '@nestjs/common';
import { UserService } from './user.service';
import { UserController } from './user.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UserEntity } from './entities/user.entity';

@Module({
  imports: [TypeOrmModule.forFeature([UserEntity])],
  controllers: [UserController],
  providers: [UserService],
  exports: [UserService],
})
export class UserModule {}
```

create-user.dto.ts

```
import { IsEmail, Length } from 'class-validator';
import { UniqueOnDatabase } from '../../auth/validations/UniqueValidation';
import { UserEntity } from '../entities/user.entity';

export class CreateUserDto {
  @Length(3)
  fullName: string;

  @IsEmail(undefined, { message: 'Неверная почта' })
  @UniqueOnDatabase(UserEntity, {
    message: 'Пользователь с такой почтой уже существует',
  })
  email: string;

  @Length(6, 32, { message: 'Пароль должен быть не менее 6 символов' })
  password?: string;
}
```

login-user.dto.ts

```
import { IsEmail, Length } from 'class-validator';

export class LoginUserDto {
  @IsEmail(undefined, { message: 'Неверная почта' })
  email: string;

  @Length(6, 32, { message: 'Пароль должен быть не менее 6 символов' })
  password?: string;
}
```

search-user.dto.ts

```
export class SearchUserDto {
  email?: string;
  fullName?: string;
  limit?: number;
  take?: number;
}
```

update-user.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateUserDto } from './create-user.dto';

export class UpdateUserDto extends PartialType(CreateUserDto) {}
```

user.decorator.ts

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common';
import { UserEntity } from '../user/entities/user.entity';

export const User = createParamDecorator(
    (_: unknown, ctx: ExecutionContext): UserEntity => {
        const request = ctx.switchToHttp().getRequest();
        return request.user.id;
    },
);
```

user.controller.ts

```
import {
  Controller,
  Get,
  Body,
  Patch,
  Param,
  UseGuards,
  Request,
  Query,
} from '@nestjs/common';
import { UserService } from './user.service';
import { UpdateUserDto } from './dto/update-user.dto';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { SearchUserDto } from './dto/search-user.dto';

@Controller('users')
export class UserController {
  constructor(private readonly userService: UserService) {}

  @Get()
  findAll() {
    return this.userService.findAll();
  }

  @UseGuards(JwtAuthGuard)
  @Get('me')
  getProfile(@Request() req) {
    return this.userService.findById(req.user.id);
  }

  @UseGuards(JwtAuthGuard)
  @Patch('me')
  update(@Request() req, @Body() updateUserDto: UpdateUserDto) {
    return this.userService.update(+req.user.id, updateUserDto);
  }

  @Get('search')
  search(@Query() dto: SearchUserDto) {
    return this.userService.search(dto);
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.userService.findById(+id);
  }
}
```

user.entity.ts

```
import {
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  OneToMany,
} from 'typeorm';
import { CommentEntity } from '../../comment/entities/comment.entity';

@Entity('users')
export class UserEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  fullName: string;

  @Column({
    unique: true,
  })
  email: string;

  @OneToMany(() => CommentEntity, (comment) => comment.user, {
    eager: false,
    nullable: true,
  })
  comments: CommentEntity[];

  @Column({ nullable: true })
  password?: string;

  @CreateDateColumn({ type: 'timestamp' })
  createdAt: Date;

  @UpdateDateColumn({ type: 'timestamp' })
  updatedAt: Date;
}
```

POST

post.service.ts

```
import {
  ForbiddenException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { CreatePostDto } from './dto/create-post.dto';
import { UpdatePostDto } from './dto/update-post.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { PostEntity } from './entities/post.entity';
import { SearchPostDto } from './dto/search-post.dto';

@Injectable()
export class PostService {
  constructor(
    @InjectRepository(PostEntity)
    private repository: Repository<PostEntity>,
  ) {}

  findAll() {
    return this.repository.find({
      order: {
        createdAt: 'DESC',
      },
    });
  }

  async popular() {
    const qb = this.repository.createQueryBuilder();

    qb.orderBy('views', 'DESC');
    qb.limit(10);

    const [items, total] = await qb.getManyAndCount();

    return {
      items,
      total,
    };
  }

  async search(dto: SearchPostDto) {
    const qb = this.repository.createQueryBuilder('p');

    qb.leftJoinAndSelect('p.user', 'user');
```

```
  qb.limit(dto.limit || 0);
  qb.take(dto.take || 10);

  if (dto.views) {
    qb.orderBy('views', dto.views);
  }

  if (dto.body) {
    qb.andWhere(`p.body LIKE :body`);
  }

  if (dto.title) {
    qb.andWhere(`p.title LIKE :title`);
  }

  if (dto.tag) {
    qb.andWhere(`p.tags LIKE :tag`);
  }

  qb.setParameters({
    title: `%${dto.title}%`,
    body: `%${dto.body}%`,
    tag: `%${dto.tag}%`,
    views: dto.views || '',
  });

  const [items, total] = await qb.getManyAndCount();

  return { items, total };
}

async findOne(id: number) {
  await this.repository
    .createQueryBuilder('posts')
    .whereInIds(id)
    .update()
    .set({
      views: () => 'views + 1',
    })
    .execute();

  return this.repository.findOne(id);
}

create(dto: CreatePostDto, userId: number) {
  const firstParagraph = dto.body.find((obj) => obj.type === 'paragraph')
    ?.data?.text;
```

```
    return this.repository.save({
      title: dto.title,
      body: dto.body,
      tags: dto.tags,
      user: { id: userId },
      description: firstParagraph || '',
    });
  }

  async update(id: number, dto: UpdatePostDto, userId: number) {
    const find = await this.repository.findOne(+id);

    if (!find) {
      throw new NotFoundException('Статья не найдена');
    }

    const firstParagraph = dto.body.find((obj) => obj.type === 'paragraph')
      ?.data?.text;

    return this.repository.update(id, {
      title: dto.title,
      body: dto.body,
      tags: dto.tags,
      user: { id: userId },
      description: firstParagraph || '',
    });
  }

  async remove(id: number, userId: number) {
    const find = await this.repository.findOne(+id);

    if (!find) {
      throw new NotFoundException('Статья не найдена');
    }

    if (find.user.id !== userId) {
      throw new ForbiddenException('Нет доступа к этой статье!');
    }

    return this.repository.delete(id);
  }
}
```

post.module.ts

```
import { Module } from '@nestjs/common';
import { PostService } from './post.service';
import { PostController } from './post.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { PostEntity } from './entities/post.entity';

@Module({
  imports: [TypeOrmModule.forFeature([PostEntity])],
  controllers: [PostController],
  providers: [PostService],
})
export class PostModule {}
```

post.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  Query,
  UseGuards,
} from '@nestjs/common';
import { PostService } from './post.service';
import { CreatePostDto } from './dto/create-post.dto';
import { UpdatePostDto } from './dto/update-post.dto';
import { SearchPostDto } from './dto/search-post.dto';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { User } from '../decorators/user.decorator';
import { UserEntity } from '../user/entities/user.entity';

@Controller('posts')
export class PostController {
  constructor(private readonly postService: PostService) {}

  @UseGuards(JwtAuthGuard)
  @Post()
  create(@User() userId: number, @Body() createPostDto: CreatePostDto) {
    return this.postService.create(createPostDto, userId);
  }

  @UseGuards(JwtAuthGuard)
  @Patch(':id')
```

```
  update(
    @User() userId: number,
    @Param('id') id: string,
    @Body() updatePostDto: UpdatePostDto,
  ) {
    return this.postService.update(+id, updatePostDto, userId);
  }

  @UseGuards(JwtAuthGuard)
  @Delete(':id')
  remove(@User() userId: number, @Param('id') id: string) {
    return this.postService.remove(+id, userId);
  }

  @Get()
  findAll() {
    return this.postService.findAll();
  }

  @Get('/popular')
  getPopularPosts() {
    return this.postService.popular();
  }

  @Get('/search')
  searchPosts(@Query() dto: SearchPostDto) {
    return this.postService.search(dto);
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.postService.findOne(+id);
  }
}
```

post.entity.ts

```
import {
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  ManyToOne,
} from 'typeorm';
import { OutputBlockData } from '../dto/create-post.dto';
import { UserEntity } from '../../user/entities/user.entity';

@Entity('posts')
export class PostEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @Column({ type: 'json' })
  body: OutputBlockData[];

  @Column()
  description: string;

  @ManyToOne(() => UserEntity, { eager: true })
  user: UserEntity;

  @Column({
    default: 0,
  })
  views: number;

  @Column({ nullable: true })
  tags?: string;

  @CreateDateColumn({ type: 'timestamp' })
  createdAt: Date;

  @UpdateDateColumn({ type: 'timestamp' })
  updatedAt: Date;
}
```

create-post.dto.ts

```
import { IsArray, IsOptional, IsString } from 'class-validator';

export interface OutputBlockData {
  id?: string;
  type: any;
  data: any;
}

export class CreatePostDto {
  @IsString()
  title: string;

  @IsArray()
  body: OutputBlockData[];

  @IsOptional()
  @IsArray()
  tags: string;
}
```

search-post.dto.ts

```
export class SearchPostDto {
  title?: string;
  body?: string;
  views?: 'DESC' | 'ASC';
  limit?: number;
  take?: number;
  tag?: string;
}
```

update-post.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreatePostDto } from './create-post.dto';

export class UpdatePostDto extends PartialType(CreatePostDto) {}
```

COMMENT

comments-service.ts

```
import { Injectable } from '@nestjs/common';
import { CreateCommentDto } from './dto/create-comment.dto';
import { UpdateCommentDto } from './dto/update-comment.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { CommentEntity } from './entities/comment.entity';

@Injectable()
export class CommentService {
 constructor(
  @InjectRepository(CommentEntity)
  private repository: Repository<CommentEntity>,
 ) {}

 async create(dto: CreateCommentDto, userId: number) {
  const comment = await this.repository.save({
   text: dto.text,
   post: { id: dto.postId },
   user: { id: userId },
  });

  return this.repository.findOne({ id: comment.id }, { relations: ['user'] });
 }

 async findAll(postId: number) {
  const qb = this.repository.createQueryBuilder('c');

  if (postId) {
   qb.where('c.postId = :postId', { postId });
  }

  const arr = await qb
   .leftJoinAndSelect('c.post', 'post')
   .leftJoinAndSelect('c.user', 'user')
   .getMany();

  return arr.map((obj) => {
   return {
    ...obj,
    post: { id: obj.post.id, title: obj.post.title },
   };
  });
 }

 findOne(id: number) {
```

```
    return this.repository.findOne(id);
  }

  update(id: number, dto: UpdateCommentDto) {
    return this.repository.update(id, dto);
  }

  remove(id: number) {
    return this.repository.delete(id);
  }
}
```

comment-module.ts

```
import { Module } from '@nestjs/common';
import { CommentService } from './comment.service';
import { CommentController } from './comment.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { CommentEntity } from './entities/comment.entity';

@Module({
  imports: [TypeOrmModule.forFeature([CommentEntity])],
  controllers: [CommentController],
  providers: [CommentService],
})
export class CommentModule {}
```

comment-controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
  Query,
} from '@nestjs/common';
import { CommentService } from './comment.service';
import { CreateCommentDto } from './dto/create-comment.dto';
import { UpdateCommentDto } from './dto/update-comment.dto';
import { User } from '../decorators/user.decorator';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';

@Controller('comments')
export class CommentController {
  constructor(private readonly commentService: CommentService) {}

  @Post()
  @UseGuards(JwtAuthGuard)
  create(@Body() createCommentDto: CreateCommentDto, @User() userId: number) {
    return this.commentService.create(createCommentDto, userId);
  }

  @Get()
  findAll(@Query() query: { postId?: string }) {
    return this.commentService.findAll(+query.postId);
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.commentService.findOne(+id);
  }

  @Patch(':id')
  update(@Param('id') id: string, @Body() updateCommentDto: UpdateCommentDto) {
    return this.commentService.update(+id, updateCommentDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.commentService.remove(+id);
  }
}
```

create-comment.dto.ts

```
import { IsNotEmpty } from 'class-validator';

export class CreateCommentDto {
  @IsNotEmpty()
  text: string;

  @IsNotEmpty()
  postId: number;
}
```

update-comments.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateCommentDto } from './create-comment.dto';

export class UpdateCommentDto extends PartialType(CreateCommentDto) {}
```

comment.entity.ts

```
import {
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  ManyToOne,
  JoinColumn,
} from 'typeorm';
import { UserEntity } from '../../user/entities/user.entity';
import { PostEntity } from '../../post/entities/post.entity';

@Entity('comments')
export class CommentEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  text: string;

  @ManyToOne(() => UserEntity, {
    nullable: false,
  })
  @JoinColumn({ name: 'userId' })
  user: UserEntity;

  @ManyToOne(() => PostEntity, {
    nullable: false,
  })
  @JoinColumn({ name: 'postId' })
  post: PostEntity;

  @CreateDateColumn({ type: 'timestamp' })
  createdAt: Date;

  @UpdateDateColumn({ type: 'timestamp' })
  updatedAt: Date;
}
```

AUTH

auth.service.ts

```
import { ForbiddenException, Injectable } from '@nestjs/common';
import { UserService } from '../user/user.service';
import { UserEntity } from '../user/entities/user.entity';
import { JwtService } from '@nestjs/jwt';
import { CreateUserDto } from '../user/dto/create-user.dto';

@Injectable()
export class AuthService {
  constructor(
    private userService: UserService,
    private jwtService: JwtService,
  ) {}

  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.userService.findByCond({
      email,
      password,
    });
    if (user && user.password === password) {
      const { password, ...result } = user;
      return result;
    }
    return null;
  }

  generateJwtToken(data: { id: number; email: string }) {
    const payload = { email: data.email, sub: data.id };
    return this.jwtService.sign(payload);
  }

  async login(user: UserEntity) {
    const { password, ...userData } = user;
    return {
      ...userData,
      token: this.generateJwtToken(userData),
    };
  }

  async register(dto: CreateUserDto) {
    try {
      const { password, ...userData } = await this.userService.create({
        email: dto.email,
        fullName: dto.fullName,
        password: dto.password,
      });
```

```
      return {
        ...userData,
        token: this.generateJwtToken(userData),
      };
    } catch (err) {
      throw new ForbiddenException('Ошибка при регистрации');
    }
  }
}
```

auth.module.ts

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { UserModule } from '../user/user.module';
import { PassportModule } from '@nestjs/passport';
import { LocalStrategy } from './strategies/local.strategy';
import { JwtModule } from '@nestjs/jwt';
import { JwtStrategy } from './strategies/jwt.strategy';

@Module({
  imports: [
    UserModule,
    PassportModule,
    JwtModule.register({
      //шифруем токен
      secret: 'test',
      signOptions: { expiresIn: '30d' }, //через 30 дней токен будет не валидным
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, LocalStrategy, JwtStrategy],
})
export class AuthModule {}
```

auth.controller.ts

```
import {
  Controller,
  Post,
  UseGuards,
  Request,
  Get,
  Body,
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { LocalAuthGuard } from './guards/local-auth.guard';
import { JwtAuthGuard } from './guards/jwt-auth.guard';
import { CreateUserDto } from '../user/dto/create-user.dto';

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @UseGuards(LocalAuthGuard)
  @Post('login')
  async login(@Request() req) {
    return this.authService.login(req.user);
  }

  @Post('register')
  register(@Body() dto: CreateUserDto) {
    return this.authService.register(dto);
  }
}
```

jwt-auth.guard.ts

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {}
```

local-auth.guard.ts

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class LocalAuthGuard extends AuthGuard('local') {}
```

jwt.strategy.ts

```
import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { UserService } from '../../user/user.service';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(private readonly userService: UserService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: 'test',
    });
  }
  async validate(payload: { sub: number; email: string }) {
    const data = { id: payload.sub, email: payload.email };
    const user = await this.userService.findByCond(data);
    if (!user) {
      throw new UnauthorizedException('Нет доступа к этой странице');
    }
    return {
      id: user.id,
      email: user.email,
    };
  }
}
```

local.strategy.ts

```
import { Strategy } from 'passport-local';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { AuthService } from '../auth.service';

@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private authService: AuthService) {
    super({ usernameField: 'email' }); //, passwordField: 'password'
  }

  async validate(email: string, password: string): Promise<any> {
    const user = await this.authService.validateUser(email, password);
    if (!user) {
      throw new UnauthorizedException('Неверный логин или пароль');
    }
    return user;}}
```

UniqueValidation.ts

```
import {
  registerDecorator,
  ValidationArguments,
  ValidationOptions,
  ValidatorConstraint,
  ValidatorConstraintInterface,
} from 'class-validator';
import { getManager } from 'typeorm';
import { UserEntity } from '../../user/entities/user.entity';

@ValidatorConstraint({ async: true })
export class UniqueOnDatabaseExistConstraint
  implements ValidatorConstraintInterface
{
  async validate(value: any, args: ValidationArguments) {
    const entity = args.object[`class_entity_${args.property}`];
    return getManager()
      .count(entity, { [args.property]: value })
      .then((count) => count < 1);
  }
}

export function UniqueOnDatabase(
  entity: any,
  validationOptions?: ValidationOptions,
) {
  validationOptions = {
    ...{ message: '$value already exists. Choose another.' },
    ...validationOptions,
  };
  return function (object: any, propertyName: string) {
    object[`class_entity_${propertyName}`] = entity;
    registerDecorator({
      target: object.constructor,
      propertyName: propertyName,
      options: validationOptions,
      constraints: [],
      validator: UniqueOnDatabaseExistConstraint,
    });
  };
}
```

Репозитории

SummerJournal-frontend: https://github.com/KozinAlexandr/SummerJournal-frontend

SummerJournal-backend: https://github.com/KozinAlexandr/SummerJournal-backend

Заключение

О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ
научно-исследовательской работы
(получение первичных навыков научно-исследовательской работы)

студента ____Козин Александр Александрович_____
(ФИО студента)

За время прохождения учебной практики научно-исследовательской работы (получение первичных навыков научно-исследовательской работы) мероприятия, запланированные в индивидуальном плане, выполнены полностью.

В процессе выполнения поставленной практической задачи «клон российского интернет-издания и агрегатора новостей TJ (tjournal).» студент продемонстрировал умение практического применения теоретических знаний, полученных в курсе программистских дисциплин, способность целенаправленного поиска необходимой информации в информационных сетях, проявил высокий уровень самостоятельности.

По окончании практики руководителем был заслушан отчет бакалавра по результатам проведенных мероприятий. Работа заслуживает оценки «_____».

Руководитель учебной практики
доцент кафедры информационных технологий
факультета компьютерных технологий
и прикладной математики, к.ф.-м.н., доцент _____ Лукащик Е.П.

Сведения о прохождении инструктажа по ознакомлению с требованиями охраны труда, технике безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка

Предприятие Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Кубанский государственный университет»
Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

Студент Козин Александр Александрович (2002 г.)
(ФИО, год рождения)
Дата 06 июля 2022 г.

1. Инструктаж по требованиям охраны труда

Провел доцент кафедры Лукащик Е.П. _____
(должность, ФИО сотрудника, проводившего инструктаж, подпись)

Прослушал Козин А.А. _____
(ФИО, подпись студента)

2. Инструктаж по технике безопасности

Провел доцент кафедры Лукащик Е.П. _____
(должность, ФИО сотрудника, проводившего инструктаж, подпись)

Прослушал Козин А.А. _____
(ФИО, подпись студента)

3. Инструктаж по пожарной безопасности

Провел доцент кафедры Лукащик Е.П. _____
(должность, ФИО сотрудника, проводившего инструктаж, подпись)

Прослушал Козин А.А. _____
(ФИО, подпись студента)

4. Инструктаж по правилам внутреннего трудового распорядка

Провел доцент кафедры Лукащик Е.П. _____
(должность, ФИО сотрудника, проводившего инструктаж, подпись)

Прослушал Козин А.А. _____
(ФИО, подпись студента)