Перечень сокращений и обозначений

В настоящем отчёте используются следующие сокращения и обозначения:

БД – база данных др. - другие

Изд-во. – издательство

ИС – информационная система источн. – источник

ПК – персональный компьютер ПО – программное обеспечение прил. – приложение

рис. – рисунок с. – страница табл. – таблица т.д. – так далее

ТЗ – техническое задание

ТО – техническое обеспечение учеб. – учебное

Реферат

Отчёт 19с., 1ч., 2 рис., 1 табл., 4 источ., 2 прил.

РАЗРАБОТКА, REACT, NODEJS, ПРОБЛЕМЫ РАЗРАБОТКИ, КЛИЕНТ-СЕРВЕР.

Цель работы: получение профессиональных знаний и умений по профессиональной деятельности, накопление данных для выпускной квалификационной работы.

Производственная практика осуществлялась в компании Ronas It.

В ходе выполнения работы было реализовано клиент-серверное приложение в соответствии с техническим заданием.

**СОДЕРЖАНИЕ**

[Термины и определения 4](#_Toc98236422)

[ВВЕДЕНИЕ 5](#_Toc98236423)

[1 Анализ рабочей области 6](#_Toc98236424)

[1.1 Изучение требований 6](#_Toc98236425)

[1.2 Методология Agile 6](#_Toc98236426)

[2 Постановка задачи 7](#_Toc98236427)

[2.1 Входные данные 7](#_Toc98236428)

[2.2 Макет 7](#_Toc98236429)

[3 Решение задачи 9](#_Toc98236430)

[3.1 Выбор языка программирования и фреймворков 9](#_Toc98236431)

[3.2 Хранилище данных 9](#_Toc98236432)

[3.3 Документация 9](#_Toc98236433)

[3.4 Разработка 10](#_Toc98236434)

[3.5 Результат разработки 12](#_Toc98236435)

[Заключение 13](#_Toc98236436)

[Приложение А 14](#_Toc98236437)

[Приложение Б 15](#_Toc98236438)

[Список использованных источников 19](#_Toc98236439)

# Термины и определения

В настоящем отчёте по производственной практике применяются следующие термины с соответствующими определениями:

Информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы, которые обеспечивают и распространяют информацию.

Программное обеспечение (ПО) – программа или множество программ, используемых для управления компьютером.

Техническое задание (ТЗ) – документ или несколько документов, определяющих цель, структуру, свойства и методы какого-либо проекта, и исключающие двусмысленное толкование различными исполнителями.

Техническое обеспечение (ТО) – это комплекс технических средств, предназначенных для работы информационной системы, а также соответствующая документация на эти средства и технологические процессы.

React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

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

NestJs - это фреймворк для создания эффективных масштабируемых Node. js серверные приложения, созданные с использованием TypeScript и полностью поддерживающий его.

Typescript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

# ВВЕДЕНИЕ

В данном документе представлен отчёт, демонстрирующий качество прохождения студентом производственной практики и оформленный согласно ГОСТ 7.32 – 2017.

Данная практика была проведена с целью получения профессиональных умений и опыта профессиональной деятельности.

Исследование проводилось в компании по разработке, поддержке и тестирование ПО – Ronas It. Изначально компания была организована как маленькая веб-студия в 2007 году, а штат состоял из нескольких человек.

На данный момент Ronas It имеет в своём составе 2 офиса, ведущих деятельность в разных городах (Омск, Краснодар). Штат в свою очередь составляет более 100 человек. В основном компания принимает заказы от иностранных заказчиков и оказывает услуги от создания дизайна, до поддержки готового продукта.

# 1 Анализ рабочей области

# 1.1 Изучение требований

В начале выполнения любого проекта первой стадией является изучение требований, согласование с заказчиком и ответственным за этот проект сотрудником. По результатам анализа поставленных задач принимается решение о методологии разработки, сроки выполнения и распределение задач между сотрудниками.

# 1.2 Методология Agile

Методика Agile — это итеративный подход к управлению проектами и разработке ПО, позволяющий командам ускорить доставку ценности клиентам и избежать лишних проблем. Вместо того чтобы выпускать весь продукт целиком, agile-команда выполняет работу в рамках небольших, но удобных инкрементов. Требования, планы и результаты постоянно проходят проверку на актуальность, благодаря чему команды могут быстро реагировать на изменения.

Ниже приведены графики сравнения двух популярных методологий: Agile и водопадной

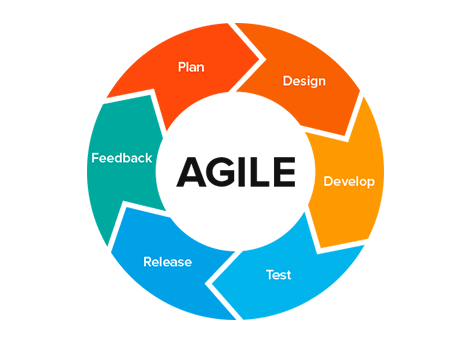


Рисунок 1 – Структура гибкой методологии

# 2 Постановка задачи

# 2.1 Входные данные

К входным данным относится поставленная руководителем задача, без первоначального макета (реализация определенной механики).

Авторизация пользователя – процесс, требующий взаимодействия клиента, сервера и базы данных. Схема взаимодействия приложения с сервером представлена на рисунке 2:

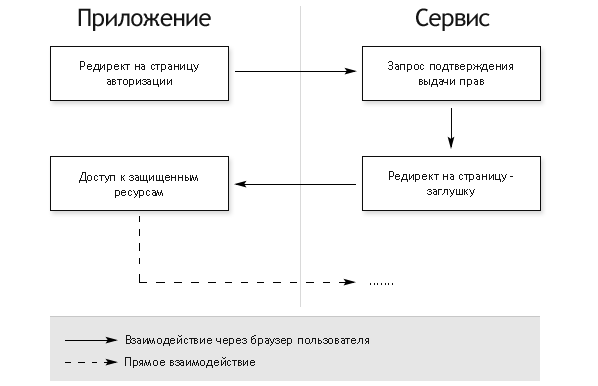


Рисунок 2 – Процесс взаимодействия клиента с сервером

# 2.2 Макет

Форма авторизации включает в себя основные поля для имени пользователя и пароля. Также важна проверка введенных данных на правильность (Все обязательные поля должны быть заполнены). Сами формы представлены на рисунках 3 и 4.

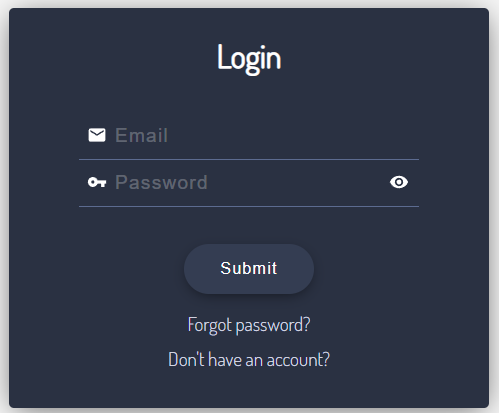


Рисунок 3 – Форма авторизации

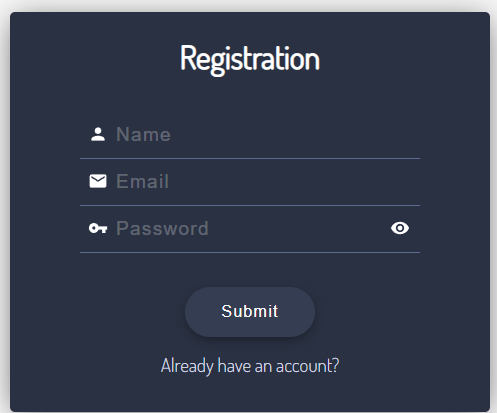


Рисунок 4 – Форма регистрации

# 3 Решение задачи

# 3.1 Выбор языка программирования и фреймворков

Выбранным языком программирования для выполнение поставленной задачи стал Typescript. Причиной тому является строгая типизация данных, современность и политика компании. Основой для создания стала библиотека React, обладающая компонентным подходом и своей универсальностью. Для серверной части используется фреймворк NestJs, полностью поддерживающий Typescript. Плюсом данного фреймворка является микросерверный подход, удобство модернизации.

# 3.2 Хранилище данных

Для хранения данных используется NoSQL база данных под управлением Mongodb, не требующая описания схемы таблицы и использующая JSON-подобные документы для хранения. Плюсом такого подхода является более быстрое написание кода и типичные для веб-разработчиков запросы.



Рисунок 5 – Схема пользователя

# 3.3 Документация

При создание серверной части приложения обязательным пунктом является написание документации с указанием возможных запросов и соответствующим их ответам. Для написания документация использовалась библиотека SwaggerApi, позволяющая указывать используемые поля и их значения в коде, в месте их объявления. Преимуществом использования данной библиотеки является стилизовання документация, работающая по указанному адресу, в нашем случае на /api/docs.

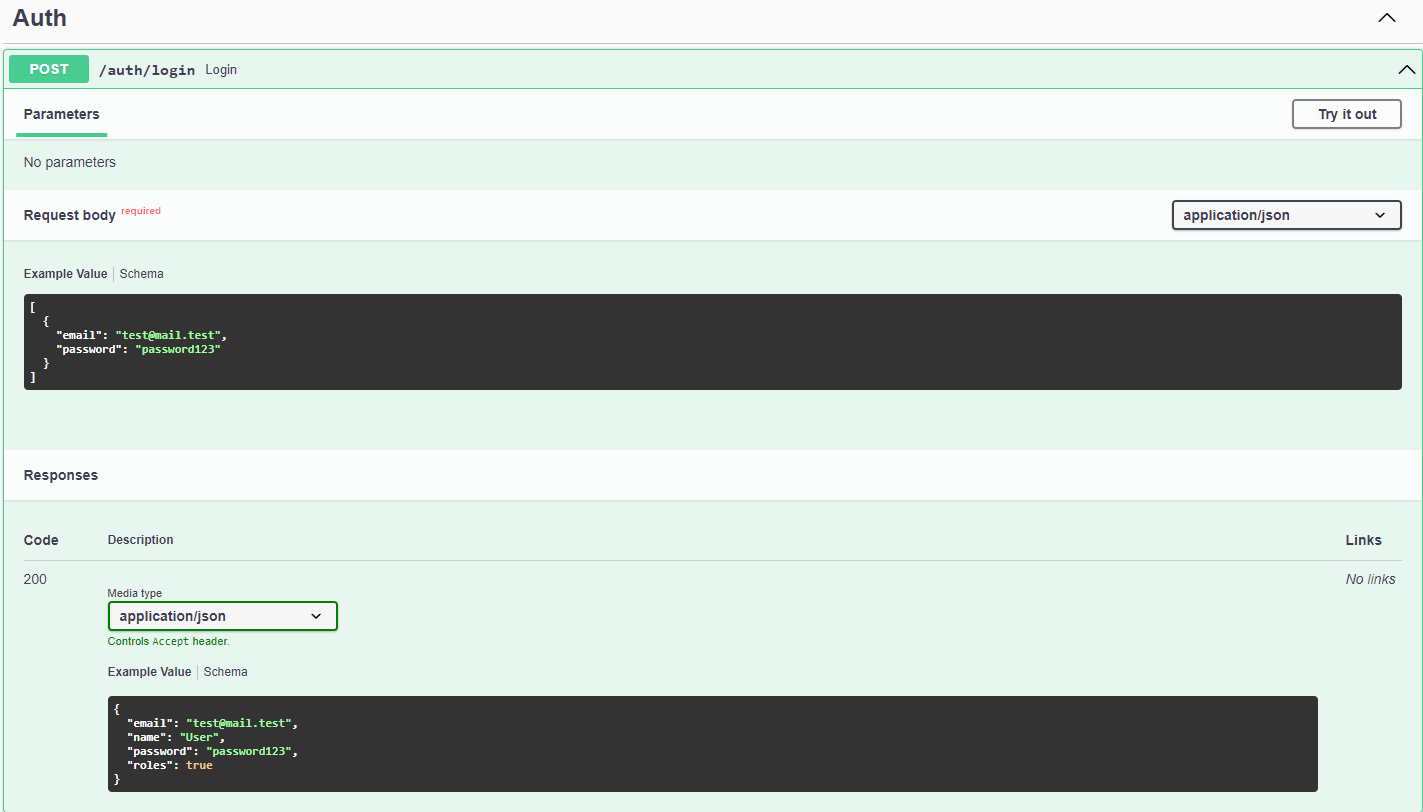


Рисунок 6 – Пример документации

# 3.4 Разработка

Процесс разработки занял большую часть времени, включая изучение новых технологий. При написании клиент-серверного приложения в первую очередь создавался сервер, принимающий запросы и на его основе клиентская часть.

При разработке клиентской части приложения используется методология БЭМ (Блок элемент модификатор), разработанная компанией Yandex. Основными понятиями такого подхода являются:

* Блок — это независимый интерфейсный компонент. Блок может быть простым или составным (содержать другие блоки). При создании блока нужно обеспечивать возможность его использования в любом месте web-страницы, а также повторения в том же самом месте страницы (родительском элементе). Блок должен включать в себя всю реализацию, необходимую для представления части интерфейса, которую он выражает.
* Элемент — это составная часть блока. Элементы контекстно-зависимы: они имеют смысл только в рамках своего блока. Элемент — не обязательная составляющая блока, небольшие блоки обходятся без элементов.
* Модификатор — это свойство блока или элемента, задающее изменения в их внешнем виде или поведении. Модификатор может быть булевым (например, button\_big) или парой ключ-значение (например, menu\_type\_bullet, menu\_type\_numbers). У блока или элемента может быть несколько модификаторов одновременно.

Также были задействованы принципы объекто-ориентированного программирования – принципы SOLID. Содержат в себе:

* Single responsibility — принцип единственной ответственности;
* Open-closed — принцип открытости / закрытости;
* Liskov substitution — принцип подстановки Барбары Лисков;
* Interface segregation — принцип разделения интерфейса;
* Dependency inversion — принцип инверсии зависимостей.

Принцип единственной обязанности / ответственности (single responsibility principle / SRP) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности.

Принцип открытости / закрытости (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.

Принцип подстановки Барбары Лисков (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом».

Принцип разделения интерфейса (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип инверсии зависимостей (dependency inversion principle / DIP) модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций.

Для управления состоянием приложения была подключена библиотека Redux, позволяющая хранить источник состояний в одном месте. В дополнение была использована библиотека Redux-Saga, позволяющая контролировать и обрабатывать все запросы пользователей в более удобном и понятном формате.

Для изменения элемента select использовалась библиотека с открытым исходным кодом React-Select. Имеющая достаточно большую популярность среди сообщества React разработчиков.

Для обращения с формами использовалась библиотека Formik, позволяющая взаимодействовать с формами в более простом формате. Также содержит в себе предустановленные функции для обработки запросов формы.

# 3.5 Результат разработки

Итоговым продуктом является готовое клиент-серверное приложение, адаптирующееся к мобильному представлению и полностью соответствующая заданному макету. Была реализована возможность авторизации с помощью социальных сетей, и первоначальная настройка профиля пользователя.

# Заключение

В ходе прохождения производственной практики был проведён анализ рабочего места, создание web-приложения и его тестирование. Были получены проблемы при разработке. По одной из выявленных проблем был предложено и исполнено решение, который заключается в использовании аналитической дополнительных библиотек для контроля состояний приложения. Помимо данной работы была также сформирована тема выпускной квалификационной работы бакалавра.

# Приложение А

(обязательное)

Код сервиса авторизации пользователя

import { HttpException, HttpStatus, Injectable, UnauthorizedException } from '@nestjs/common';

import { JwtService } from '@nestjs/jwt';

import { CreateUserDto } from 'src/users/dto/create-user.dto';

import { UsersService } from 'src/users/users.service';

import \* as bcrypt from 'bcrypt';

import { AuthUserDto } from 'src/users/dto/auth-user.dto';

@Injectable()

export class AuthService {

    constructor(private usersService: UsersService,

        private jwtService: JwtService) { }

    async login(userDto: AuthUserDto) {

        const user = await this.validateUser(userDto)

        return this.generateToken(user)

    }

    async registration(userDto: CreateUserDto) {

        const candidate = await this.usersService.getUserByEmail(userDto.email)

        if (candidate) {

            throw new HttpException('User exist', HttpStatus.BAD\_REQUEST)

        }

        const hashPassword = await bcrypt.hash(userDto.password, 5)

        const user = await this.usersService.createUser({ ...userDto, password: hashPassword })

        return this.generateToken(user)

    }

    private async generateToken(user) {

        const payload = {

            email: user.email,

            name: user.name,

            roles: user.roles

        }

        return {

            token: this.jwtService.sign(payload)

        }

    }

    private async validateUser(userDto: AuthUserDto) {

        const user = await this.usersService.getUserByEmail(userDto.email);

        if (user) {

            const passwordEquals = await bcrypt.compare(userDto.password, user.password);

            if (passwordEquals) {

                return user

            }

        }

        throw new UnauthorizedException({ message: "Wrong name or password" })

    }

}

# Приложение Б

(обязательное)

Код страницы авторизации

import { ErrorMessage, Field, Form, Formik, validateYupSchema } from 'formik'

import React, { useState } from 'react'

import { MdMail, MdVpnKey, MdRemoveRedEye } from "react-icons/md"

import { useNavigate } from 'react-router-dom'

import { history } from '../../\_helpers/history'

import { authService } from '../../\_services/auth.service'

import styles from './loginpage.module.scss'

function LoginPage() {

    let navigate = useNavigate();

    const [status, setStatus] = useState("")

    const required = (value: any) => (value ? undefined : setStatus("Email and password required"));

    const [showPassword, showPasswrodStatus] = useState(Boolean)

    return (

        <div className={styles.login}>

            <Formik

                initialValues={{ email: "", password: "" }}

                onSubmit={(values) => {

                    setStatus("")

                    authService.login(values.email, values.password)

                        .then(

                            user => {

                                navigate("/news")

                            },

                            error => {

                                setStatus(error)

                            }

                        );

                }}

            >

                <Form className={styles.form}>

                    <span className={styles.form\_header}>

                        <h1>Login</h1>

                    </span>

                    <div className={styles.input\_container}>

                        <div className={styles.input\_container\_\_field}>

                            <MdMail className={styles.input\_container\_\_icon} size='20px' />

                            <Field

                                className={`${styles.input\_\_control} ${status ? styles.error : ''}`}

                                validate={required}

                                name="email"

                                type="email"

                                placeholder="Email"

                            />

                        </div>

                        <div className={styles.input\_container\_\_field}>

                            <MdVpnKey className={styles.input\_container\_\_icon} size='20px' />

                            <Field

                                className={`${styles.input\_\_control} ${status ? styles.error : ''}`}

                                validate={required}

                                name="password"

                                type={showPassword ? 'text' : 'password'}

                                placeholder="Password"

                            />

                            <MdRemoveRedEye

                                className={`${styles.input\_container\_\_icon} ${styles.input\_container\_\_password}`}

                                size='20px'

                                onClick={() => {

                                    showPasswrodStatus(!showPassword)

                                }}

                            />

                        </div>

                    </div>

                    <button className={styles.form\_\_button} type="submit">Submit</button>

                    <div className={styles.login\_links}>

                        <span className={styles.login\_links\_\_item}>

                            <a href='#'>

                                Forgot password?

                            </a>

                        </span>

                        <span className={styles.login\_links\_\_item}>

                            <a href='/registration'>

                                Don't have an account?

                            </a>

                        </span>

                        {status ? <div className={styles.alert}>{status}</div> : <br></br>}

                    </div>

                </Form>

            </Formik>

        </div>

    )

}

export default LoginPage

import { ErrorMessage, Field, Form, Formik } from 'formik'

import React, { useState } from 'react'

import { MdMail, MdPerson, MdRemoveRedEye, MdVpnKey } from 'react-icons/md'

import { useNavigate } from 'react-router-dom'

import { authService } from '../../\_services/auth.service'

import styles from './registrationpage.module.scss'

function RegistrationPage() {

    let navigate = useNavigate();

    const [status, setStatus] = useState("")

    const required = (value: any) => (value ? undefined : setStatus("User information required"));

    const [showPassword, showPasswrodStatus] = useState(Boolean)

    return (

        <div className={styles.registration}>

            <Formik

                initialValues={{ name: "", email: "", password: "" }}

                onSubmit={(values) => {

                    setStatus("")

                    authService.registration(values.name, values.email, values.password)

                        .then(

                            user => {

                                navigate("/news")

                            },

                            error => {

                                setStatus(error)

                            }

                        );

                }}

            >

                <Form className={styles.form}>

                    <span className={styles.form\_header}>

                        <h1>Registration</h1>

                    </span>

                    <div className={styles.input\_container}>

                        <div className={styles.input\_container\_\_field}>

                            <MdPerson className={styles.input\_container\_\_icon} size='20px' />

                            <Field

                                className={`${styles.input\_\_control} ${status ? styles.error : ''}`}

                                name="name"

                                validate={required}

                                type="text"

                                placeholder="Name"

                            />

                        </div>

                        <div className={styles.input\_container\_\_field}>

                            <MdMail className={styles.input\_container\_\_icon} size='20px' />

                            <Field

                                className={`${styles.input\_\_control} ${status ? styles.error : ''}`}

                                name="email"

                                validate={required}

                                type="email"

                                placeholder="Email"

                            />

                        </div>

                        <div className={styles.input\_container\_\_field}>

                            <MdVpnKey className={styles.input\_container\_\_icon} size='20px' />

                            <Field

                                className={`${styles.input\_\_control} ${status ? styles.error : ''}`}

                                name="password"

                                validate={required}

                                type={showPassword ? 'text' : 'password'}

                                placeholder="Password"

                            />

                            <MdRemoveRedEye

                                className={`${styles.input\_container\_\_icon} ${styles.input\_container\_\_password}`}

                                size='20px'

                                onClick={() => {

                                    showPasswrodStatus(!showPassword)

                                }}

                            />

                        </div>

                    </div>

                    <button className={styles.form\_\_button} type="submit">Submit</button>

                    <div className={styles.login\_links}>

                        <span className={styles.login\_links\_\_item}>

                            <a href='/login'>

                                Already have an account?

                            </a>

                        </span>

                        {status ? <div className={styles.alert}>{status}</div> : <br></br>}

                    </div>

                </Form>

            </Formik>

        </div>

    )

}

export default RegistrationPage

# Список использованных источников

1. ГОСТ 19.201-78 Единая система программной документации. Техническое задание. Требование к содержанию и оформлению.
2. ГОСТ 7.32-2017 – Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления.
3. Официальная документация библиотеки React

<https://ru.reactjs.org/>

4) Материально-техническое обеспечение Ronas It

https://projects.ronasit.com/ronasit-docs/staff/blob/master