ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Ордена Трудового Красного Знамени
Федеральное государственное бюджетное образовательное учреждение
высшего образования
МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра «Математическая кибернетика и информационные технологии»

«Сайт для тестирования на уровень IQ»

по дисциплине «Проектный практикум»

Выполнил: Студент группы БВТ2306
Кесслер Алексей Сергеевич

Москва 2024

Содержание

Введение

Для своего проекта я выбрал тему “Сайт для тестирования на уровень IQ”. Эта идея показалась мне интересной и полезной, так как многие мои друзья и знакомые задаются вопросом об их уровне IQ. Мой сайт поможет людям узнать каким уровнем они обладают (относительно теста Ганса Айзенка)

Я выбрал эту тему, потому что хочу улучшить свои навыки в создании реактивных веб-интерфейсов, создании собственного backend-приложения с написанием API. Так же мне интересно создавать такие сайты, которые будут полезны как мне, так и другим.

Я поставил пред собой цель создать сайт для тестирования на уровень IQ, а также сделать его доступным в интернете

Для достижения цели я поставил следующие задачи:

1. Создать backend-приложение для хранения и обработки результатов тестирования, а также для отправки email писем
2. Реализовать frontend часть с адаптивным дизайном
3. Сделать сайт доступным в интернете

Выполнение работы

Архитектура приложения

В качестве архитектуры приложения я выбрал стандартную Layer архитектуру:

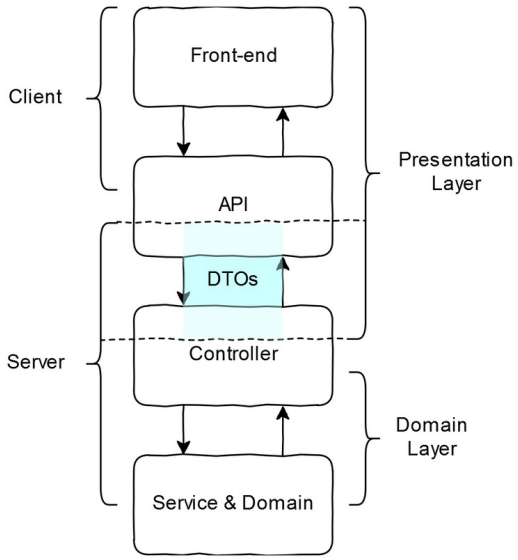

Рис. 1 - Архитекутра приложения

В качестве домена (основы, ядра приложения) будут выступать данные о результатах тестирования, которые будут находится в базе данных.

Сервисы отвечают за операции с данными и реализацией отдельного функционала приложения в backend части. Сервис и домен взаимодействуют с контроллером в двухстороннем порядке.

Контроллер обрабатывает входящие запросы.

Запросы поступают в контроллер с помощью REST API и содержат DTO (Data Transfer Object) в качестве тела запроса.

DTO используются для передачи данных между слоями приложениями.

Клиентская часть (frontend) получает и отправляет данные с помощью GET и POST API запросов.

Создание базы данных

В качестве базы данных я выбрал MongoDB. Зарегистрировался на их сайте, создал новый проект “Iq-check”, в этом проекте создал новый бесплатный кластер и получил строку для подключения к БД.

MongoDB - это нереляционная база данных, использующая для хранения данных формат BSON (очень похож на JSON), поэтому она не требует проектирования архитектуры и может разрабатываться по ходу проекта.

Создание backend-приложения

В качестве фреймворка для создания backend'а я выбрал Express.js (Node.js web Application framework) + TypeScript для типизации данных. Всё приложение будет работать на Node.js, поэтому устанавливаем Node.js v22.11.0 (в месте с Node.js установится пакетный менеджер npm).

Инициализируем проект и устанавливаем зависимости:

- Production dependencies: express, body-parser, class-transformer, class-validator, config, corse doting, express-validator, inversify, mongoose, nodemailer, reflect-metadata, ts-log, @types/cors

- Development dependencies: @types/express, @types/mongoose, @types/nodemailer, nodemon, ts-node, typescript

Я решил реализовать следующие эндпоинты для API:

- /result
  - GET: Получение результата тестирования по id
  - POST: Запись результата тестирования в БД
- /recent
  - GET: Получение последних 20 результатов тестирований
- /top
  - GET: Получение последних 3 лучших результатов тестирований за последнюю неделю
- /restore
  - POST: Восстановление результата тестирования (отправка на почту сертификата с результатами последнего тестирования)

Для всех api-роутов будет использоваться префикс /api/v1

Далее я реализовал контроллеры (каждый контроллер реализован в отдельном TypeScript файле):

- base.controller.ts - родительский класс для всех контроллеров (Приложение 1)
- result.controller.ts - (Приложение 2)
- recent.controller.ts
- top.controller.ts
- restore.controller.ts

И сервисы:

- config.service.ts - предоставляет методы для получения данных из .env файла
- database.service.ts - предоставляет методы для подключения к базе данных
- logger.service.ts - предоставляет методы для логгирования
- mail.service.ts - предоставляет методы для отправки email писем (Приложение 3)
- result.servise.ts - предоставляет методы для записи и получения результатов тестирований (Приложение 4)

В проекте будут следующие DTO:

- result-get.dto.ts
    - ResultGetByIdDto
    - ResultGetRecentDto
    - ResultGetByEmailDto
    - ResultGetTopDto
- result-set.dto.ts
    - ResultSetDto
    - ResultRestoreDto

Также для валидации данных будут использоваться middleware (промежуточный слой приложения), а для обработки ошибок будут использоваться filters (фильтры)

- validate.middleware.ts - промежуточный слой для валидации данных
- exception.filter.ts - фильтр обработки ошибки
- https-error.class.ts - класс кастомной http ошибки

Для управления данными в базе данных используются mongoose-модели. В моем проекте она будет всего одна:

- Result.model.ts - (Приложение 5)

Все контроллеры, сервисы, middlewares, exceptionFilters, будут подключены в файле App.ts (Приложение 6)

Сам сервер будет запущен с помощью файла server.ts и в качестве приложения будет использовать контейнер App, созданный из класса, описанного в файле App.ts (Приложение 7)

Для типизации всех параметров, возвращаемых значений и тд используется TypeScript. Все интерфейсы данных я поместил в папку src/interfaces в репозитория проекта.

Backend приложение запускается командой:

- В development режиме - npm run server
- В production режиме - npm run start (перед этим необходимо сбилдить проект командой npm run build)

Создание frontend-приложения

Структура сайта будет следующей:

- Главная (Сам тест, 3 лучших результата недели, 20 последних результатов, IQ рейтинг знаменитостей, информация об IQ тестировании, шапка и подвал с навигацией.
- Контакты
- Условия использования
- Политика конфиденциальности
- Восстановление результата
- Сертификат с результатами тестирования
- Страница не найдена (ошибка 404)

Поскольку почти все страницы - это статичная информация, то рассмотрим только главную страницу.

Я разработал адаптивный дизайн в такой стилистике:

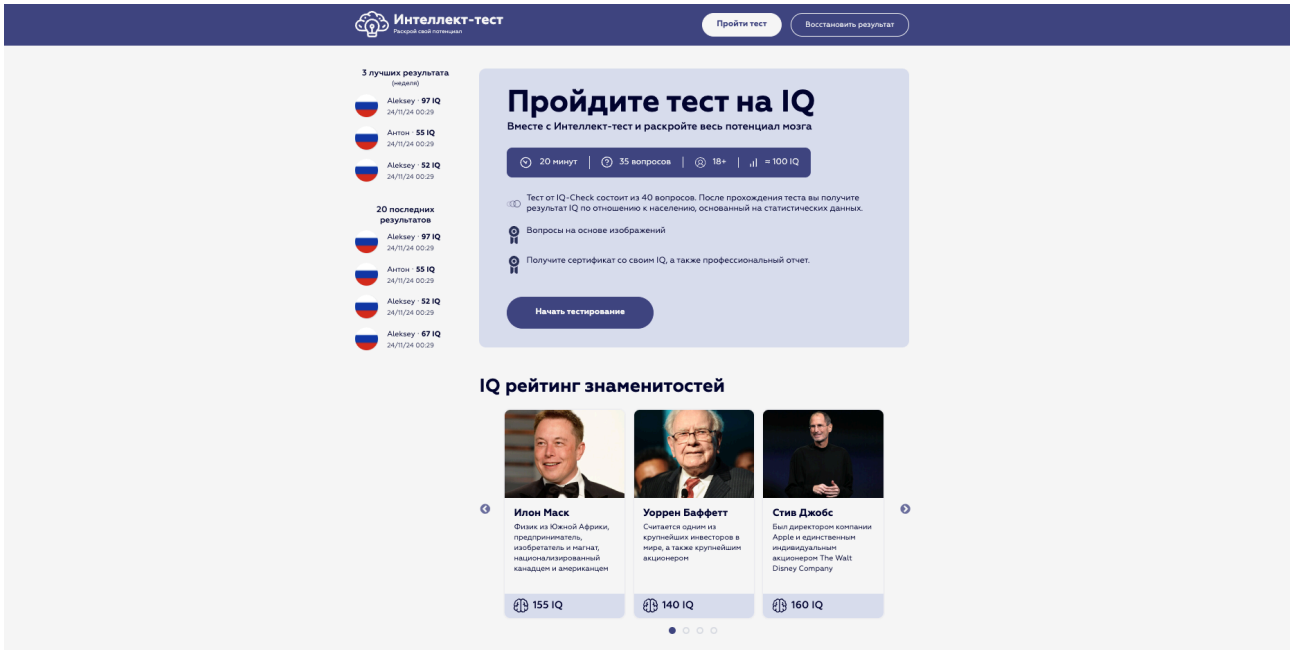

Рис. 2 - Дизайн главной страницы

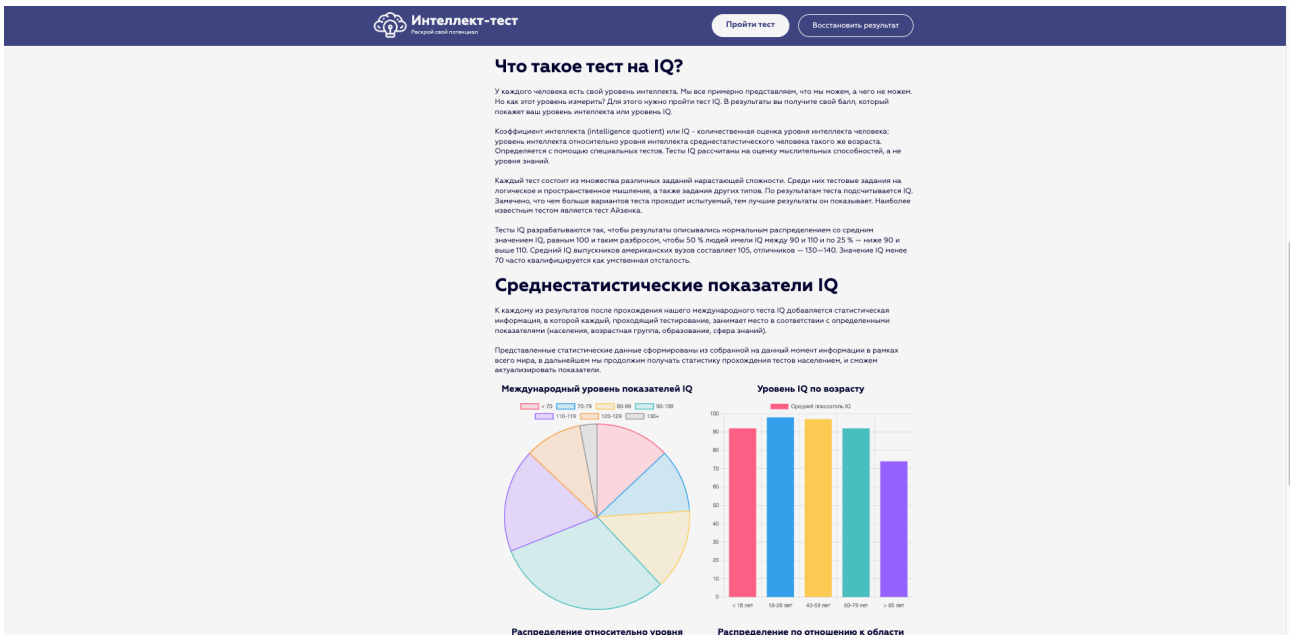

Рис. 3 - Дизайн главной страницы

Frontend я создавал с помощью NextJS, Redux Toolkit + RTKQuery, TypeScript.

Я выбрал именно NextJS, а не ReactJS, из-за возможности рендеринга на стороне сервера (SSR), так как очень важна SEO оптимизация сайта (так как трафик я собираюсь привлекать из поисковых систем так как по запросам “iq тест”, “тест на интеллект” суммарно 300 тысяч запросов ежемесечно), которая практически полностью отсутствует в SPA приложениях.

В качестве подхода к написанию стилей я использовал CSS Modules.

В качестве Стейт-менеджера я использовал Redux Toolkit + RTKQuery, так как для меня это самая удобная связка.

Для рисования графиков и диаграмм я использовал библиотеку Chart.js

Разберем стор приложения. Он имеет в себе два редьюсера (Приложение 8):

- resultApi.reducer - отвечает за работу с API backend-приложения, имеет следующие методы (Приложение 9):
    - addResult
    - getResult

    - getResults
    - getRecentResults
    - getTopResults
    - restoreResult
- testSlice.reducer - отвечает за работу самого теста на IQ, имеет следующие экшены (Приложение 10):
    - nextSlide
    - startTest
    - addResultPoints
    - resetTest
    - finishTest

Часть верстки будет повторяться на каждой страницы, поэтому вынесем её в отдельный Layout, который содержит в себе шапку, контейнер для главного конвента страницы и подвал (Приложение 11)

На главной странице используется условный рендеринг (Приложение 12). Если поднят флаг из стейта isTestStarted, то отображаем компонент с самим тестом, если нет, то отображаем информацию о тесте и кнопку “начать тест”. На главную страницу должны подтягиваться данные о 3х последних лучших результатах за неделю и 20 недавних результатов. Экспортируем из файла функцию getServerSideProps, в которой отправляем GET API запросы к нашему backend’у, и передаем полученные данные как props’ы для рендера страницы. В итоге сервер отдаст страницу с уже отрендеренными данными и не придется делать запрос с клиента.

Компонент Test тоже использует условный рендеринг (Приложение 13). Если поднят флаг isTestFinished (тест закончен), то показываем форму для заполнения результатов. Если тест не закончен, то показываем вопрос. Сам массив вопросов с картинками записан в отдельную константу. За отображение конкретного вопроса ответственно свойство currentSlide в стейте.

Следовательно когда это свойство меняется, то компонент Test рендерится заново с обновленными данными - это принцип реактивности, используемый в React.js (и как следствие в Next.js).

Сам компонент вопроса Question (Приложение 14) рендерит картинку вопроса и 6 вариантов ответа. При нажатии на один из вариантов ответа мы диспатчим экшены в наш стейт. finishTest Если пользователь ответил на последний вопрос, иначе nextSlide, и в любом случае диспатчим addResultPoints - добавляем баллы к результату пользователя (добавляем 0, если ответ неверный).

Если тест закончился, то рендерим компонент SendResultForm (Приложение 15) с формой отправки результата на сервер (пользователь должен заполнить информацию о себе, прежде чем получит результат). После отправки формы вызываем триггер мутации, который возвращается при вызове щука useAddResultMutation. Этот и другие хуки возвращаются из редьюсера api.reducer. На сервер уходит POST /result запрос с информацией о результате тестирования. В ответ возвращается ID этого результата в базе данных. Этот ID представляет из себя хэш из 32 символов.

Дальше пользователя переадресовывает на страницу /certificate/{ID}, на котором отображаются результаты тестирования. Таким образом пользователь получает постоянную ссылку на результаты теста. Также эта ссылка отправляется на почту, которую указал пользователь.

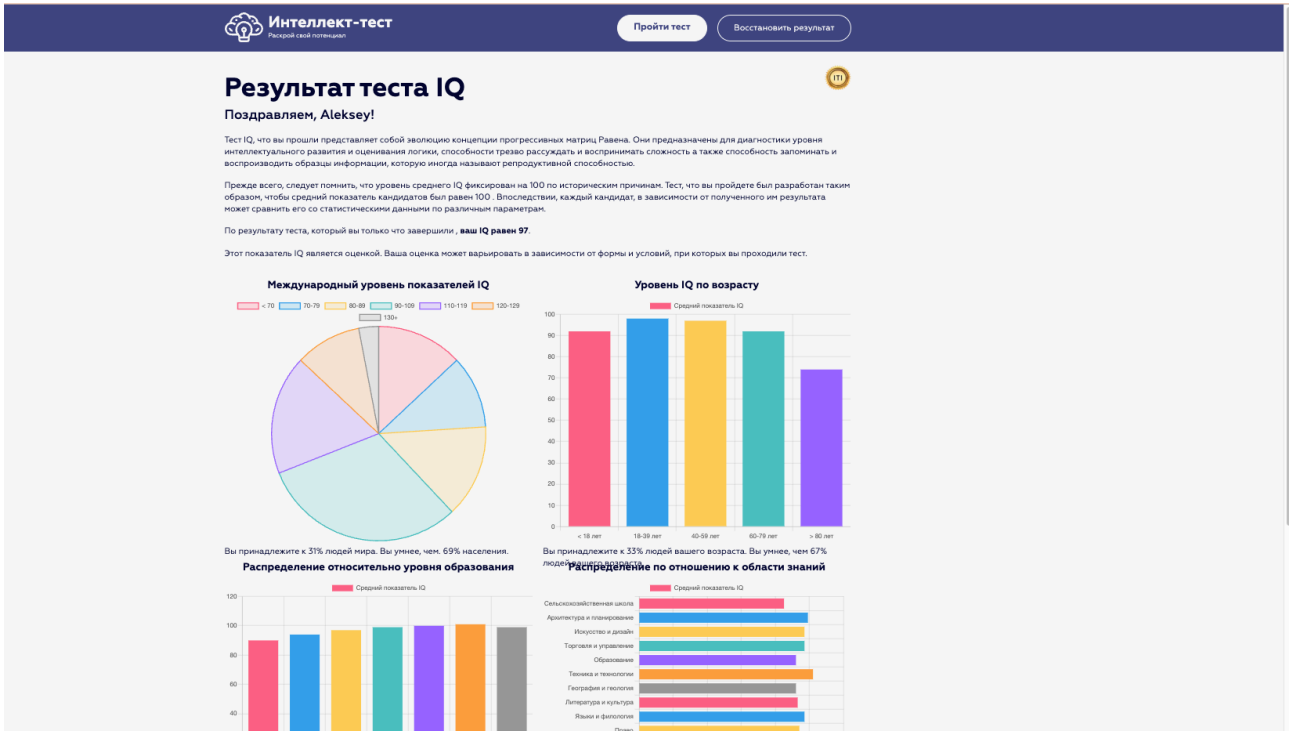

Рис. 4 - Сертификат с результатами теста

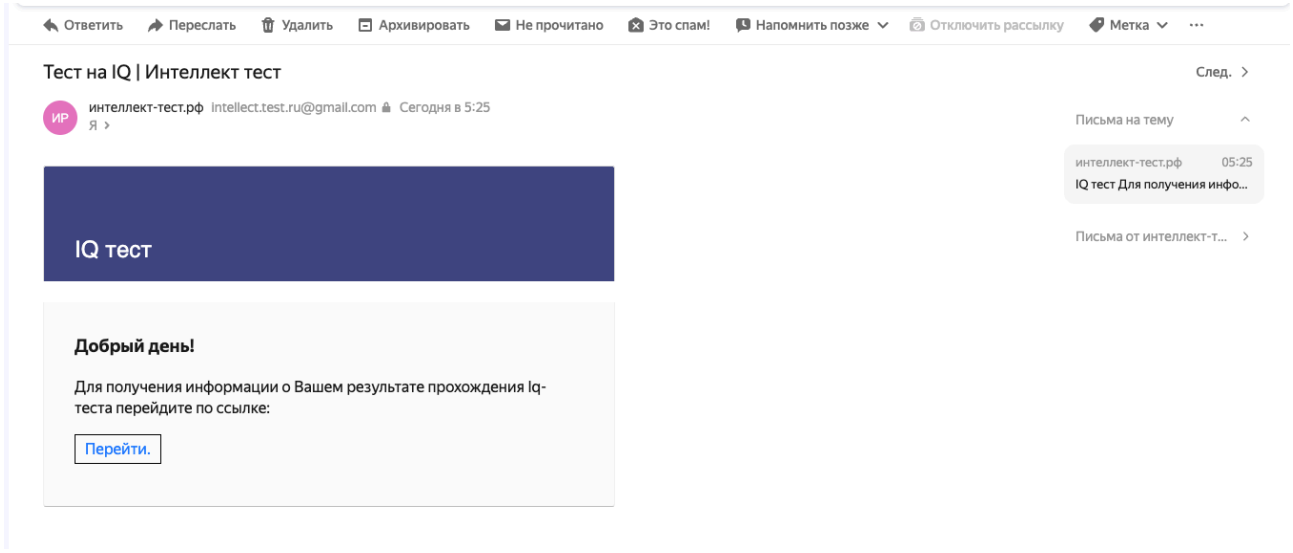

Рис. 5 - Письмо с ссылкой на сертификат

Если пользователь потерял ссылку на сертификат, то он может её восстановить. Для этого создана страница “Восстановить результаты”. На ней находится форма, в которую пользователь вводит свой email, после чего на указанный адрес приходит письмо с ссылкой на последние результаты

тестирования для пользователя с этим адресом электронной почты.

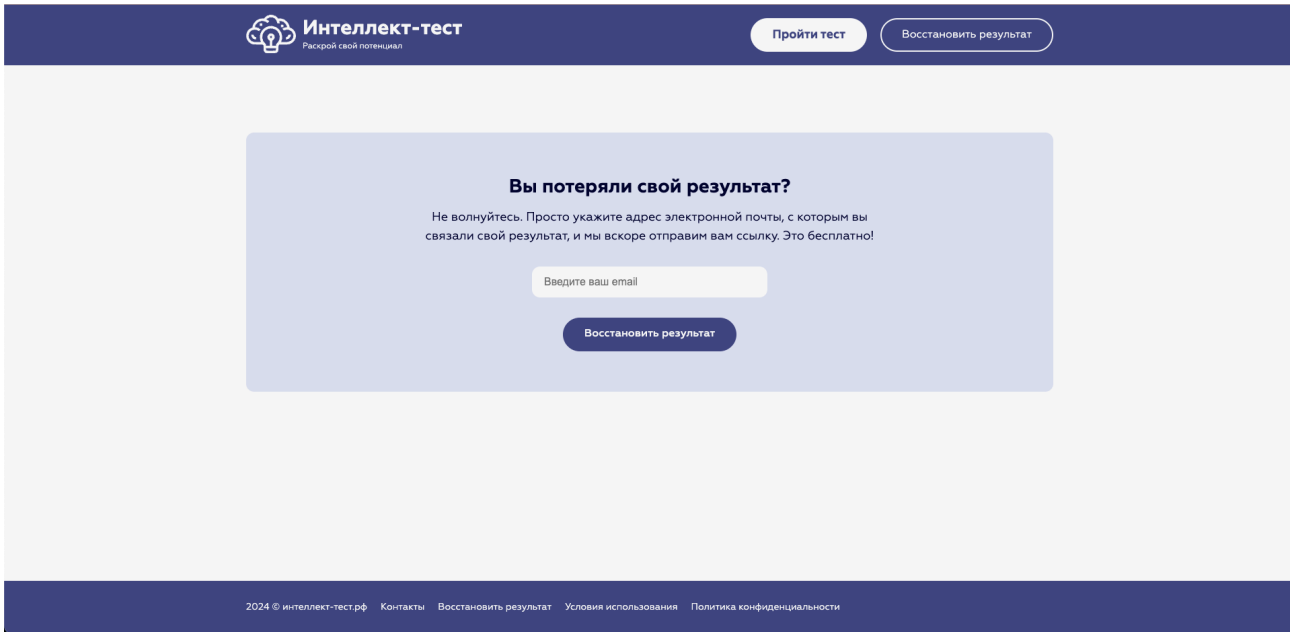

Рис. 6 - Страница “Восстановить результат”

На остальных страницах (Контакты, Политика конфиденциальности и тд) размещена статичная информация.

Также на сайт я установил счетчик Яндекс.Метрики для того чтобы собирать аналитику по посещениям моего сайта.

Развертывание приложения на VDS сервере

Я купил домен iq-check.ru и арендовал VDS сервер. Для VDS сервера я выпустил SSL сертификат с помощью certbot для того, чтобы сайт работал по HTTPS. В ресурсных записях домена я создал запись A-типа и указал в ней ip адрес моего VDS сервера - 80.249.150.24

Установим git на сервер - sudo app install git

Выпустим SSH сертификат для добавления его в свой профиль GitHub для того чтобы складировать репозиторий:

- ssh-keygen -t rsa
- cat /root/.ssh/id_rsa.pub

Клонируем репозиторий - git clone git@github.com:LightninG8/MTUCI.git

Устанавливаем Node.js v22.11.0 на сервер - sudo apt install nodejs и npm - sudo apt install npm

Установим все зависимости в оба проекта - npm install

Создадим .env файл со всеми ключами и конфиденциальной информацией - nano .env

В роли демона будем использовать npm пакет pm2, установим его глобально - npm install pm2 -g

Запустим backend - pm2 start “npm run start” —name=“server”

И frontend (в своей папке) - pm2 start “npm run start” —name=“client”

В качестве веб-сервера будет использовать Nginx. Настроим Конфиг (Приложение 16)

Обновим nginx - sudo servise nginx restart

Теперь сайт работает в онлайн.

Заключение

В ходе выполнения работы я создал веб-приложение для тестирования на уровень IQ, познакомился с адаптивным дизайном, улучшил свои навыки работы с Next.js и Express.js.

Ссылка на GitHub - https://github.com/LightninG8/MTUCI/tree/main/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9%D0%9F%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D1%83%D0%BC

Ссылка на сайт - https://iq-check.ru

Приложения

Приложение 1 - base.controller.ts

```
import { Response, Router } from 'express';
import { IControllerRoute, ILogger } from "interfaces";
import { injectable } from 'inversify';
import 'reflect-metadata';

@injectable()
export abstract class BaseController {
  private readonly _router: Router;

  constructor(private logger: ILogger) {
    this._router = Router();
  }

  get router() {
    return this._router;
  }

  public send<T>(res: Response, code: number, message: T)
{
    res.type('application/json');

    return res.status(code).json(message);
  }

  public ok<T>(res: Response, message: T) {
    return this.send<T>(res, 200, message);
  }
```

```
  public created(res: Response) {
    return res.sendStatus(201);
  }

  protected bindRoutes(routes: IControllerRoute[]) {
    for(const route of routes) {
      this.logger.log(`[${route.method.toUpperCase()}] $
{route.path}`);

      const middleware = route.middlewares?.map((m) =>
m.execute.bind(m));
      const handler = route.func.bind(this);
      const pipeline = middleware ? [...middleware,
handler] : handler;

      this.router[route.method](route.path, pipeline);
    }
  }
}
```

Приложение 2 - result.controller.ts

```
import { BaseController } from "./base.controller";
import { NextFunction, Request, Response } from
"express";
import { injectable, inject } from 'inversify';
import { TYPES } from "../types";
import { ILogger, IResultService } from "../interfaces";
import 'reflect-metadata';
import { IRecentController } from "../interfaces";
import { ValidateMiddleware } from "../middlewares";
```

```
import { ResultGetRecentDto } from "../dto";

@injectable()
export class RecentController extends BaseController
implements IRecentController{
  constructor(
    @inject(TYPES.ILogger) private loggerService:
ILogger,
    @inject(TYPES.IResultService) private resultService:
IResultService,
  ) {
    super(loggerService);

    this.bindRoutes([
      {
        path: '/recent',
        method: 'get',
        func: this.getRecent,
        middlewares: [new
ValidateMiddleware(ResultGetRecentDto)]
      }
    ])
  }
  async getRecent(req: Request, res: Response, next:
NextFunction) {
    try {
      const recentResults = await
this.resultService.getRecent(req.query as unknown as
ResultGetRecentDto);
```

```
      if (!recentResults?.length) {
        return this.send(res, 401, {
          message: `Последние ${req.query.limit} не
найдены`
        })
      }

      this.ok(res, {
        message: 'Результаты найдены',
        data: recentResults
      })
    } catch (e) {
      return this.send(res, 500, {
        message: `Ошибка сервера. Повтроите запрос
позднее. ${e}`
      })
    }
  }
}
```

Приложение 3 - Result.model.ts

```
import { Schema, model } from "mongoose";

const ResultSchema = new Schema({
  email: {
    type: String,
    required: true,
    unique: false
  },
  name: {
    type: String,
    required: true
  },
```

```
  iq: {
    type: Number,
    required: true
  },
  gender: {
    type: String,
  },
  yearOfBirth: {
    type: String,
  },
  educationType: {
    type: String,
    required: true
  },
  educationLevel: {
    type: String,
    required: true
  },
  countryCode: {
    type: String,
    required: true
  }
}, { timestamps: true });

export const ResultModel = model('Result', ResultSchema);
```

Приложение 4 - mail.servise.ts

```
import { IMailService, ILogger, IConfigService } from
'../interfaces';
import { inject, injectable } from 'inversify';
import { TYPES } from '../types';
import { createTestAccount, createTransport, TestAccount,
Transporter } from 'nodemailer';


@injectable()
export class MailService implements IMailService {
  transporter: Transporter


  constructor(
```

```
    @inject(TYPES.IConfigService) private configService:
IConfigService,
    @inject(TYPES.ILogger) private logger: ILogger
  ) {}

  async connect() {
    try {
      this.transporter = createTransport({
        host: 'smtp.gmail.com',
        port: 465,
        secure: true, // true for 465, false for other
ports 587
        auth: {
          user: this.configService.get('MAIL_LOGIN'),
          pass: this.configService.get('MAIL_SECURE_KEY')
        }
      });

      this.logger.log('[Mail] Почтовый сервис подключен')
    } catch (e) {
      this.logger.error('[Mail] Ошибка подключения
почтового сервиса')
    }

  }

  async send(email: string, resultId: string) {
    const ADDRESS = process.env.NODE_ENV == 'development'
? 'http://localhost:3000' : 'https://интеллект-тест.рф';
```

```
    await this.transporter.sendMail({
      from: `"интеллект-тест.рф" $
{this.configService.get('MAIL_LOGIN')}`,
      to: email,
      subject: 'Тест на IQ | Интеллект тест',
      text: 'Для получения информации о Вашем результате
прохождения IQ-теста перейдите по ссылке',
      html: `
        <table border="0" cellspacing="0" cellpadding="0"
style="max-width:600px">
            <tbody>
            <tr>
                <td>
                    <table bgcolor="#3E4480" width="100%"
border="0" cellspacing="0" cellpadding="0" style="min-
width:332px;max-width:600px;border:1px solid
#e0e0e0;border-bottom:0;border-top-left-
radius:3px;border-top-right-radius:3px">
                        <tbody>
                        <tr><td height="72"
colspan="3"></td></tr>
                        <tr>
                            <td width="32"></td>
                            <td style="font-
family:Roboto-Regular,Helvetica,Arial,sans-serif;font-
size:24px;color:#ffffff;line-height:1.25">
                                IQ тест
                            </td>
                            <td width="32"></td>
                        </tr>
```

```
                        <tr><td height="18px"
colspan="3"></td></tr>
                        </tbody>
                    </table>
                </td>
            </tr>
            <tr>
                <td>
                     <table bgcolor="#FAFAFA"
width="100%" border="0" cellspacing="0" cellpadding="0"
style="min-width:332px;max-width:600px;border:1px solid
#f0f0f0;border-bottom:1px solid #c0c0c0;border-
top:0;border-bottom-left-radius:3px;border-bottom-right-
radius:3px">
                        <tbody>
                        <tr height="16px">
                            <td width="32px"
rowspan="3"></td>
                            <td></td>
                            <td width="32px"
rowspan="3"></td>
                        </tr>
                        <tr>
                            <td>
                                <h3>Добрый день!</h3>
                                <p>
                                    Для получения
информации о Вашем результате прохождения Iq-теста
перейдите по ссылке:
                                    <br>
```

```
                                        <br>
                                        <a style="border:1px
solid #000000;padding:5px 10px;text-decoration:none"
href="${ADDRESS}/certificate/${resultId}">Перейти.</a>
                                    </p>
                                </td>
                            </tr>
                            <tr height="32px"></tr>
                            </tbody>
                        </table>
                    </td>
                </tr>
                <tr height="16px"></tr>
                </tbody>
            </table>`
        }, (error) => {
          if (error) {
            this.logger.error(`Не получилось отправить
письмо: ${error}`);
          } else {
            this.logger.log(`Письмо доставлено на адрес $
{email}`);
          }
        });
      }
}
```

Приложение 5 - result.service.ts

```
import { IResultService } from '../interfaces';
import { injectable } from 'inversify';
```

```
import { ResultGetByIdDto, ResultGetByEmailDto,
ResultGetRecentDto, ResultSetDto, ResultGetTopDto } from
'../dto';
import { ResultModel } from '../models';

@injectable()
export class ResultService implements IResultService {

  async set(body: ResultSetDto) {
    try {
      const result = new ResultModel(body);

      await result.save();

      return result;
    } catch (e) {
      return null;
    }

  }
  async getResult(body: any) {
    try {
      const result = await ResultModel.find(body);

      return result;
    } catch (e) {
      return null;
    }
  }
```

```
  async getResultById(body: ResultGetByIdDto) {
    try {
      const result = await ResultModel.findOne(body);

      return result;
    } catch (e) {
      return null;
    }
  }

  async getResultByEmail(body: ResultGetByEmailDto) {
    try {
      const result = (await
ResultModel.find(body).sort({_id: -1}).limit(1))[0];

      return result;
    } catch (e) {
      return null;
    }
  }

  async getRecent(body: ResultGetRecentDto) {
    try {
      const recentResults = await
ResultModel.find().sort({_id: -1}).limit(+body.limit);

      return recentResults;
    } catch (e) {
      return null;
    }
```

```
  }

  async getTop(body: ResultGetTopDto) {
    try {
      const daysAgoTimestamp = Date.now() - 1000 * 60 *
60 * 24 * +body.days;

      const recentResults = await
ResultModel.find({createdAt: {$gte:
daysAgoTimestamp}}).sort({iq: -1}).limit(+body.limit);

      return recentResults;
    } catch (e) {
      return null;
    }
  }
}
```

Приложение 6 - App.ts

```
import express, { Express } from 'express';
import { Server } from 'http';
import { injectable, inject } from 'inversify';
import { TYPES } from './types';
import { IConfigService, IDatabaseService,
IExceptionFilter, ILogger, IMailService,
```

```
IRecentController, IRestoreController, IResultController,
ITopController } from './interfaces';
import 'reflect-metadata';
import { json } from 'body-parser';
import cors from 'cors';

@injectable()
export class App {
  app: Express;
  server: Server;
  port: number;

  constructor (
    @inject(TYPES.ILogger) private logger: ILogger,

    @inject(TYPES.IResultController) private
resultController: IResultController,
    @inject(TYPES.IRecentController) private
recentController: IRecentController,
    @inject(TYPES.ITopController) private topController:
ITopController,
    @inject(TYPES.IRestoreController) private
restoreController: IRestoreController,


    @inject(TYPES.IExceptionFilter) private
exceptionFilter: IExceptionFilter,
    @inject(TYPES.IDatabaseService) private
databaseService: IDatabaseService,
```

```
    @inject(TYPES.IMailService) private mailService:
IMailService,


  ) {
    this.app = express();
    this.port = 8080;
  };
  useRoutes() {
    this.app.use('/api/v1',
this.resultController.router);
    this.app.use('/api/v1',
this.recentController.router);
    this.app.use('/api/v1', this.topController.router);
    this.app.use('/api/v1',
this.restoreController.router);

  };

  useExceptionFilters() {

this.app.use(this.exceptionFilter.catch.bind(this.excepti
onFilter));
  };

  useMiddleware() {
    this.app.use(json());
    this.app.use(cors());

  }
```

```
  async useDatabase() {
    await this.databaseService.connect();
  }

  async useMail() {
    await this.mailService.connect();
  }

  public async init() {
    this.useMiddleware();
    this.useRoutes();
    this.useExceptionFilters();
    await this.useDatabase();
    await this.useMail();

    this.server = await this.app.listen(this.port, () =>
{
      this.logger.log(`[App] Сервер запущен на https://
localhost:${this.port}`)
    });

  };
}
```

Приложение 7 - server.ts

```
import { ConfigService, DatabaseService, LoggerService,
ResultService, MailService } from "./services";
import { App } from "./app";
```

```
import { RecentController, RestoreController,
ResultController, TopController } from './controllers';
import { ExceptionFilter } from "./exceptions";
import { Container, ContainerModule, interfaces  } from
'inversify';
import { TYPES } from "./types";
import { IConfigService, IDatabaseService,
IExceptionFilter, ILogger, IMailService,
IRecentController, IRestoreController, IResultController,
IResultService, ITopController } from "./interfaces";

export interface IBootstrapReturn {
  appContainer: Container;
  app: App;
};

export const appBindings = new ContainerModule((bind:
interfaces.Bind) => {

bind<ILogger>(TYPES.ILogger).to(LoggerService).inSingleto
nScope();

bind<IExceptionFilter>(TYPES.IExceptionFilter).to(Excepti
onFilter).inSingletonScope();


bind<IResultController>(TYPES.IResultController).to(Resul
tController).inSingletonScope();
```

```
bind<IRecentController>(TYPES.IRecentController).to(Recen
tController).inSingletonScope();

bind<ITopController>(TYPES.ITopController).to(TopControll
er).inSingletonScope();

bind<IRestoreController>(TYPES.IRestoreController).to(Res
toreController).inSingletonScope();



bind<IConfigService>(TYPES.IConfigService).to(ConfigServi
ce).inSingletonScope();

bind<IDatabaseService>(TYPES.IDatabaseService).to(Databas
eService).inSingletonScope();

bind<IResultService>(TYPES.IResultService).to(ResultServi
ce).inSingletonScope();

bind<IMailService>(TYPES.IMailService).to(MailService).in
SingletonScope();

  bind<App>(TYPES.Application).to(App);
});

const bootstrap = async (): Promise<IBootstrapReturn> =>
{
  const appContainer = new Container();
```

```
  appContainer.load(appBindings);

  const app = appContainer.get<App>(TYPES.Application);

  await app.init();

  return { appContainer, app };
}

export const boot = bootstrap();
```

Приложение 8 - store/index.ts

```
import { configureStore } from '@reduxjs/toolkit';
import { setupListeners } from '@reduxjs/toolkit/dist/
query';
import { testSlice } from 'ducks/slices';
import { resultApi } from 'ducks/api';
import { createWrapper } from 'next-redux-wrapper';


export const makeStore = () => configureStore({
  reducer: {
    [resultApi.reducerPath]: resultApi.reducer,
    [testSlice.name]: testSlice.reducer
  },
  middleware: getDefaultMiddleware =>
getDefaultMiddleware().concat(resultApi.middleware)
});
```

```
export const store = makeStore();


setupListeners(store.dispatch);

export type RootStore = ReturnType<typeof makeStore>;
export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch;

export const wrapper =
createWrapper<RootStore>(makeStore);
```

Приложение 9 - slices/testSlice.ts

```
import { createSlice } from '@reduxjs/toolkit';

interface ITestSlice {
  isTestStarted: boolean;
  currentSlide: number;
  resultPoints: number;
  isTestFinished: boolean;
}

const initialState: ITestSlice = {
  currentSlide: 0,
  isTestStarted: false,
  resultPoints: 40,
  isTestFinished: false,
};
```

```
export const testSlice = createSlice({
  name: 'test',
  initialState,
  reducers: {
    nextSlide: (state) => {
      state.currentSlide += 1;
    },
    startTest: (state) => {
      state.currentSlide = 1;
      state.isTestStarted = true
    },
    addResultPoints: (state, action) => {
      state.resultPoints += action.payload
    },
    resetTest: (state) => {
      console.log(initialState);

      state = {
        ...state
      };
    },
    finishTest: (state) => {
      state.isTestFinished = true;
    }
  }
});


export const { nextSlide, startTest, addResultPoints,
resetTest, finishTest } = testSlice.actions;
```

Приложение 10 - api/result.api.ts

```
import { createApi, fetchBaseQuery } from '@reduxjs/
toolkit/query/react';
import { COMMON_API_ADRESS } from 'commonConstants';
import { IUser } from 'models';
import { HYDRATE } from 'next-redux-wrapper';

export const resultApi = createApi({
  reducerPath: 'resultApi',
  baseQuery: fetchBaseQuery({baseUrl: `${
{COMMON_API_ADRESS}`}),
  extractRehydrationInfo(action, { reducerPath }) {
    if (action.type === HYDRATE) {
      return action.payload[reducerPath];
    }
  },
  tagTypes: [],
  endpoints: (build) => ({
    addResult: build.mutation<any, any>({
      query: (body) => ({
        url: '/result',
        method: 'POST',
        headers: {
          'Access-Control-Allow-Origin': '*'
        },
        body
      }),
```

```
}),
getResult: build.query({
  query: (_id) => ({
    url: `/result`,
    params: {
      _id
    }
  })
}),
// TODO
getResults: build.query<any, void>({
  query: () => '/results'
}),
getRecentResults: build.query({
  query: (limit) => ({
    url: `/recent`,
    params: {
      limit
    }
  })
}),
getTopResults: build.query({
  query: ({days, limit}) => ({
    url: `/top`,
    params: {
      days,
      limit
    }
  })
}),
```

```
    restoreResult: build.mutation<any, any>({
      query: (email: string) => ({
        url: '/restore',
        method: 'POST',
        headers: {
          'Access-Control-Allow-Origin': '*'
        },
        body: {
          email
        }
      }),
    })
  })
});

export const {
  useAddResultMutation,
  useGetResultQuery,
  useGetResultsQuery,
  util: { getRunningOperationPromises },
  useRestoreResultMutation
} = resultApi;


// export endpoints for use in SSR
export const {
  getResults,
  getResult,
  getRecentResults,
  getTopResults,
```

```
  restoreResult,
} = resultApi.endpoints;
```

Приложение 11 - MainLayout.ts

```
import s from './MainLayout.module.css';
import cs from 'styles/common.module.css';
import { Footer, Header } from 'components';

type MainLayoutProps = {
  className?: string;
  children?: React.ReactNode;
}

export const MainLayout: React.FC<MainLayoutProps> =
({className, children}: MainLayoutProps) => {
  return (
    <div className={s.layout}>
      <Header />
      <main className={s.main + ' ' + className}>
        {children}
      </main>
      <Footer />
    </div>
  )
};
```

Приложение 12 - index.ts

```
import type { NextPage } from 'next';
import { MainLayout } from 'containers';
import Head from 'next/head';
import cs from 'styles/common.module.css';
import s from 'styles/Home.module.css';
import { getRecentResults, getRunningOperationPromises,
getTopResults, wrapper } from 'ducks';
import { CelebritiesSlider, Sidebar, TestPreview,
Charts } from 'components';
import { useAppSelector } from 'hooks';
import { Test } from 'containers/Test';
import { COMMON_API_ADRESS } from 'commonConstants';

interface IHomeProps {
  recentResults: any;
  topResults: any;
};

const Home: NextPage<IHomeProps> = ({recentResults,
topResults}) => {
  const { isTestStarted } = useAppSelector((state) =>
state.test);

  return (
    <>
      <Head>
        <title>IQ Тест онлайн | Интеллект-тест</title>
      </Head>
      <MainLayout>
```

```
        <div className={s.home}>
          <div className={s.home__body + ' ' +
cs.container}>
            <Sidebar className={s.home__sidebar}
recentResults={recentResults} topResults={topResults}/>
            <main className={s.home__main}>
              <section className={s.home__test}>
                {!isTestStarted && <TestPreview />}
                {isTestStarted && <Test />}
              </section>
              <section className={s.home__celebrities}>
                <h2 className={s.home__title + ' ' +
cs.title}>IQ рейтинг знаменитостей</h2>
                <CelebritiesSlider
className={s.home__slider}/>
              </section>
              <section className={s.section__text}>
                <h2 className={s.home__title + ' ' +
cs.title}>Что такое тест на IQ?</h2>
                <p className={cs.container__paragraph}>
                  У каждого человека есть свой уровень
интеллекта. Мы все примерно представляем, что мы можем, а
чего не можем. Но как этот уровень измерить? Для этого
нужно пройти тест IQ. В результаты вы получите свой балл,
который покажет ваш уровень интеллекта или уровень IQ.
                </p>
                <p className={cs.container__paragraph}>
                  Коэффициент интеллекта (intelligence
quotient) или IQ - количественная оценка уровня
интеллекта человека: уровень интеллекта относительно
```

уровня интеллекта среднестатистического человека такого же возраста. Определяется с помощью специальных тестов. Тесты IQ рассчитаны на оценку мыслительных способностей, а не уровня знаний.

Каждый тест состоит из множества различных заданий нарастающей сложности. Среди них тестовые задания на логическое и пространственное мышление, а также задания других типов. По результатам теста подсчитывается IQ. Замечено, что чем больше вариантов теста проходит испытуемый, тем лучшие результаты он показывает. Наиболее известным тестом является тест Айзенка.

Тесты IQ разрабатываются так, чтобы результаты описывались нормальным распределением со средним значением IQ, равным 100 и таким разбросом, чтобы 50 % людей имели IQ между 90 и 110 и по 25 % — ниже 90 и выше 110. Средний IQ выпускников американских вузов составляет 105, отличников — 130—140. Значение IQ менее 70 часто квалифицируется как умственная отсталость.

</section>

<section className={s.home__average}>

<h2 className={s.home__title + ' ' + cs.title}>Среднестатистические показатели IQ</h2>

К каждому из результатов после прохождения нашего

```
международного теста IQ добавляется статистическая
информация, в которой каждый, проходящий тестирование,
занимает место в соответствии с определенными
показателями (населения, возрастная группа, образование,
сфера знаний).</p>
                <p className={cs.container__paragraph}
>Представленные статистические данные сформированы из
собранной на данный момент информации в рамках всего
мира, в дальнейшем мы продолжим получать статистику
прохождения тестов населением, и сможем актуализировать
показатели.</p>
                <Charts className={s.home__charts} />
              </section>

            </main>
          </div>
        </div>
      </MainLayout>
    </>

  );
}


export const getServerSideProps =
wrapper.getServerSideProps((store) => async () => {

    const recentResults = await fetch(`${
{COMMON_API_ADRESS}/recent?limit=20`).then((data) =>
data.json())
```

```
    const topResults = await fetch(`${COMMON_API_ADRESS}/
recent?days=7&limit=3`).then((data) => data.json())

    return {
      props: {
        recentResults: recentResults?.data || [],
        topResults: topResults?.data || []
      },
    };
  }
);


export default Home
```

Приложение 13 - Test.tsx

```
import { useDispatch } from 'react-redux';
import { useAppSelector } from 'hooks';
import { questionsList } from './questionsList';
import { Question, SendResultForm } from './components';

interface ITestProps {

};


export const Test: React.FC<ITestProps> = ({}:
ITestProps) => {
```

```
  const dispatch = useDispatch();
  const { currentSlide, isTestFinished } =
useAppSelector((state) => state.test);

  return (
    <>
      { !isTestFinished && (
        <Question slide={questionsList[currentSlide -
1]}/>
      )}
      { isTestFinished && (
        <SendResultForm />
      )}


    </>
  );
}
```

Приложение 14 - Question.tsx

```
import s from './Questioin.module.css';

import { IQuestion, questionsListLength } from
'containers/Test/questionsList';
import Image from 'next/image';
import { useAppDispatch, useAppSelector } from 'hooks';
import { nextSlide, addResultPoints, finishTest } from
'ducks';
```

```
interface IQuestionProps {
  slide: IQuestion;
};


export const Question: React.FC<IQuestionProps> =
({slide}: IQuestionProps) => {
  const dispatch = useAppDispatch();

  const { currentSlide } = useAppSelector((state) =>
state.test);

  const onClick = (iqPoints: number) => {
    if (currentSlide == questionsListLength) {
      dispatch(finishTest())
    } else {
      dispatch(nextSlide());
    }

    dispatch(addResultPoints(iqPoints));
  }
  return (
    <div className={s.slide + ' ' + s.slide__body}>
      <div className={s.slide__title}
>{slide?.question?.text}</div>
      <div className={s.slide__question}>
        <Image src={slide?.question?.image}
alt='question' loading='eager'/>
      </div>
      <div className={s.slide__answers}>
```

```
        {slide?.answers?.map((item, i) => (
          <div
            className={s.answer + ' ' + s.answer__body}
            onClick={() => onClick(item.value)}
            key={i}
          >
            <Image src={item.image} alt='answer'/>
          </div>
        ))}
      </div>
      <div className={s.slide__counter}>{currentSlide}/
{questionsListLength}</div>
    </div>
  );
}
```

Приложение 15 - SendResultForm.tsx

```
import { Button } from 'components';
import { resetTest, useAddResultMutation } from 'ducks';
import { useAppDispatch, useAppSelector } from 'hooks';
import { useForm, SubmitHandler } from 'react-hook-form';
import s from './SendResultForm.module.css';
import { useRouter } from 'next/router';
import { educationLevels, educationTypes, genderTypes }
from 'commonConstants';

interface ISendResultFormProps {}
```

```
interface IFormInput {
  name: string;
  email: string;
  gender: string;
  yearOfBirth: number;
  educationLevel: string;
  educationType: string;
}
export const SendResultForm: React.FC<
  ISendResultFormProps
> = ({}: ISendResultFormProps) => {
  const dispatch = useAppDispatch();
  const router = useRouter();

  const { resultPoints } = useAppSelector((state) =>
state.test);

  const [trigger] = useAddResultMutation();
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<IFormInput>();

  const onSubmit: SubmitHandler<IFormInput> = async
(data) => {
    await trigger({
      ...data,
      iq: resultPoints,
      countryCode: navigator.language.slice(-2)
```

```
    }).then((result: any) => {
      dispatch(resetTest());

      return result;
    }).then((result: any) => {
      window.location.href = `/certificate/$
{result.data?.data?._id}`;
    })
  };

  const renderYearOfBirthOptions = () => {
    const result = [];

    for(let i = 2010; i > 1920; i--) {
      result.push(
        <option value={i} key={i}>{i}</option>
      )
    }

    return result;
  }

  return (
    <form className={s.form}
onSubmit={handleSubmit(onSubmit)}>
      <div className={s.form__header}>
        <h2 className={s.form__title}>
          Поздравляем, вы только что закончили тест!
        </h2>
        <h3 className={s.form__subtitle}>
```

```
          Для того, чтобы издать ваш показатель и
статистики (уровень IQ,
          позиции в соответствии с возрастом, сфере
обучения и уровня
          образования), просим вас заполнить следующую
информацию:
        </h3>
      </div>
      <div className={s.form__body}>
        <label className={s.form__label + ' ' + s.label}>
          <div className={s.label__title}>Имя
пользователя :</div>
          <input
            className={s.label__input}
            type='text'
            {...register('name', { required: 'Необходимо
заполнить «Имя»', maxLength: 40 })}
            aria-invalid={errors.name ? 'true' : 'false'}
          />
          { errors.name && <p className={s.label__error}
>{errors.name.message}</p> }
        </label>
        <label className={s.form__label + ' ' + s.label}>
          <div className={s.label__title}>Email :</div>
          <input
            className={s.label__input}
            type='email'
            {...register('email', {
              required: 'Необходимо заполнить «Email»',
```

```
              pattern: /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/
g,
            })}
            aria-invalid={errors.email ? 'true' :
'false'}
          />
          { errors.email && <p className={s.label__error}
>{errors.email.message}</p> }
        </label>
        <label className={s.form__label + ' ' + s.label}>
          <div className={s.label__title}>Пол :</div>
          <select className={s.label__input}
{...register('gender')}>
            <option value=''></option>
            {genderTypes.labels.map((item) => (
              <option value={item} key={item}>{item}</
option>
            ))}
          </select>
        </label>
        <label className={s.form__label + ' ' + s.label}>
          <div className={s.label__title}>Год
рождения :</div>
          <select className={s.label__input}
{...register('yearOfBirth')}>
            <option value=''></option>
            {renderYearOfBirthOptions()}
          </select>
        </label>
        <label className={s.form__label + ' ' + s.label}>
```

```
          <div className={s.label__title}>Образование :</
div>
          <select
            className={s.label__input}
            {...register('educationType', {required:
'Необходимо выбрать «Образование»'})}
            aria-invalid={errors.educationType ? 'true' :
'false'}
          >
            <option value=''></option>
            {educationTypes.labels.map((item) => (
              <option value={item} key={item}>{item}</
option>
            ))}
          </select>
          { errors.educationType && <p
className={s.label__error}>{errors.educationType.message}
</p> }
        </label>
        <label className={s.form__label + ' ' + s.label}>
          <div className={s.label__title}>Уровень
образования :</div>
          <select
            className={s.label__input}
            {...register('educationLevel', {required:
'Необходимо выбрать «Уровень образования»'})}
            aria-invalid={errors.educationLevel ?
'true' : 'false'}
          >
            <option value=''></option>
```

```
            {educationLevels.labels.map((item) => (
              <option value={item} key={item}>{item}</
option>
            ))}
          </select>
          { errors.educationLevel && <p
className={s.label__error}
>{errors.educationLevel.message}</p> }
        </label>
        <div className={s.form__submit}>
          <Button type='submit'>Получить результат</
Button>
        </div>
      </div>
    </form>
  );
};
```

Приложение 16 - nginx конфигурация

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;
```

```
        index index.html index.htm index.nginx-
debian.html;

        server_name _;

        location / {
                try_files $uri $uri/ =404;
        }
}

server {
        root /var/www/html;

        index index.html index.htm index.nginx-
debian.html;
    server_name iq-check.ru; # managed by Certbot

         location / {
                proxy_pass http://localhost:3000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
         location /api/v1/ {
                proxy_pass http://localhost:8080;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
```

```
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/iq-check.ru/
fullchain.pem; # managed >
    ssl_certificate_key /etc/letsencrypt/live/iq-
check.ru/privkey.pem; # manage>
    include /etc/letsencrypt/options-ssl-nginx.conf; #
managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #
managed by Certbot

}

server {
    if ($host = iq-check.ru) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 ;
        listen [::]:80 ;
    server_name iq-check.ru;
    return 404; # managed by Certbot
}
```