

Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и
управления»



Отчет
Лабораторная работа № 7
По курсу «Разработка интернет приложений»

ИСПОЛНИТЕЛЬ:

Группа ИУ5-55Б

Болдин Д.А.

"16" декабря 2020 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

" " _____ 2020 г.

Москва 2020

1. Задание

Необходимо разработать систему для проведения викторин по российским городам.

Работа была разделена на несколько частей:

- Проработка дизайна
- Разработка фронтенда
- Разработка бекенда
- Деплой и дальнейший мониторинг

2. Разработка фронтенда

На стороне фронтенда в проекте используется FLUX архитектура и PWA

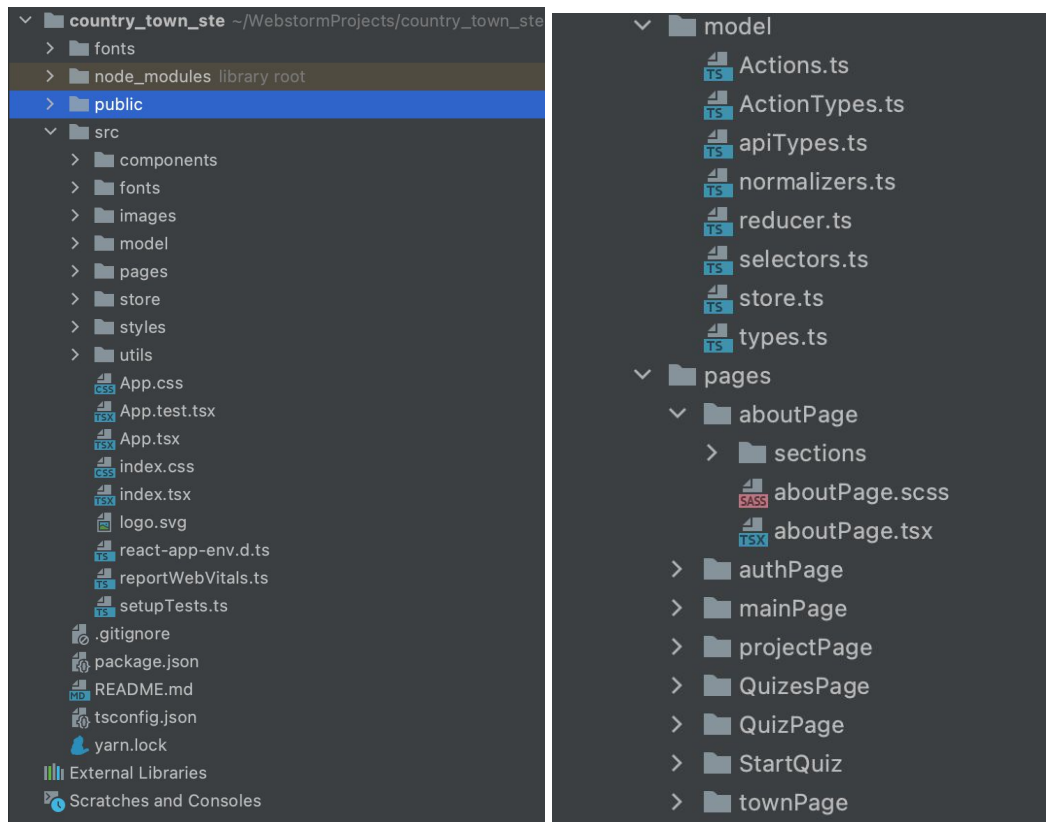
Этапы разработки фронтенда

- Сетап проекта.
- Проектирование макетов страниц в figma
- Проектирование базовых элементов интерфейса
- Проектирование хранилища веб-приложения.
- Проектирование роутинга в приложении.
- Верстка страниц и компонентов.
- Применение стилей для страниц.

Технологии, используемые в фронтенде:

- TypeScript — расширенная версия JavaScript со статической типизацией
- React - Создание пользовательских интерфейсов
- Redux - Хранилище
- HTML - Разметка
- SCSS - Стили

Структура проекта:



3. Разработка бекенда

На стороне бэкенда в проекте используется MVP архитектура.

Этапы разработки бекенда:

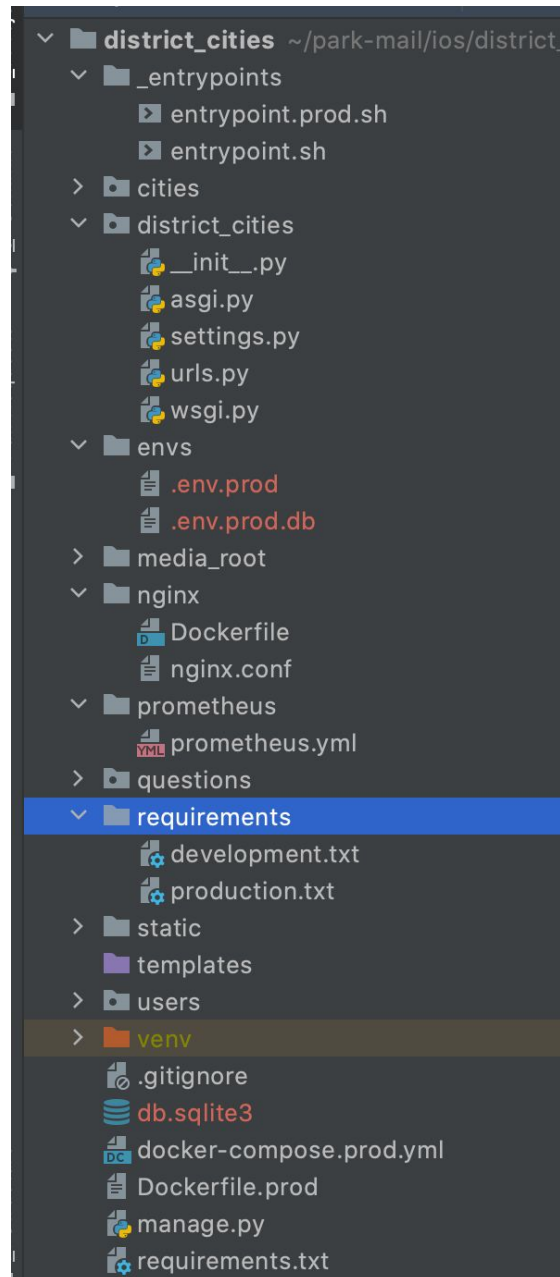
- Проектирование базы данных
- Написание моделей в Django
- Написание сериализеров для моделей с помощью Django Rest Framework
- Создание портала для администратора
- Написание view, а также добавление необходимой бизнес-логики в модели

Технологии, используемые в бекенде:

- Django
- Django Rest Framework(DRF)
- PIL(для работы с картинками)
- Django-json-widget(виджет для редактирования json в admin панели)

- Django-prometheus(мониторинг)
- Postgresql(база данных)

Структура проекта:



4. Деплой и мониторинг

После написания бекенда и фронтенда необходимо произвести деплой и настроить удобный мониторинг.

Во время деплоя были использованы следующие технологии

- Nginx для отдачи фронтенда и проксирования запроса на бекенд
- Node_exporter для отдачи информации о состоянии сервера

- UWSGI Gunicorn позволяет запустить несколько процессов с Django для ускорения работы сайта
- Prometheus собирает и хранит все данные о мониторинге из Django и Node_exporter
- Grafana используется для составления дашбордов о работе сайта на основе данных из Prometheus
- Docker-compose позволяет удобно запускать все сервисы приложения одной командой

После деплоя и настройки приложения на сервере мы выпустили сертификат для подключения по https и настроили его автоматический перевыпуск с помощью certbot.

Конфигурация docker-compose файла:

```
version: '3.3'

services:
  prometheus:
    image: prom/prometheus:v2.14.0
    volumes:
      - ./prometheus:/etc/prometheus/
    restart: always

  grafana:
    image: grafana/grafana:6.5.2
    ports:
      - 3060:3000
    restart: always

  nginx:
    build: ./nginx
    volumes:
      - static_volume:/usr/src/district_cities/static
      - static_volume:/usr/src/district_cities/media_root
    ports:
      - 127.0.0.1:9090:80
    depends_on:
      - web
    restart: on-failure

  web:
    build:
      context: .
      dockerfile: ./Dockerfile.prod
    command: gunicorn district_cities.wsgi --bind 0.0.0.0:8000
    --worker-connections=1000 --workers=3
    volumes:
      - static_volume:/usr/src/district_cities/static
      - static_volume:/usr/src/district_cities/media_root
    ports:
      - 127.0.0.1:8080:8000
```

```

env_file:
  - envs/.env.prod
depends_on:
  - db
restart: on-failure

db:
  image: postgres:12.0-alpine
  volumes:
    - postgres_data:/var/lib/postgresql/data/
  env_file:
    - envs/.env.prod.db
  restart: on-failure

node_exporter:
  image: prom/node-exporter
  ports:
    - 127.0.0.1:9100:9100
  restart: always

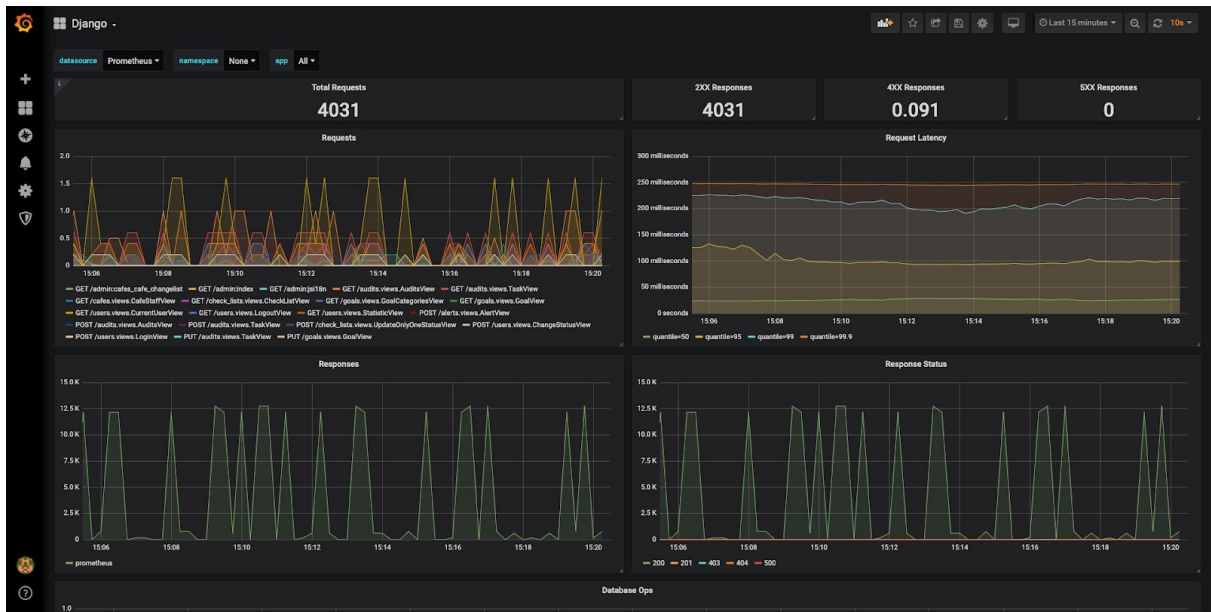
volumes:
  postgres_data:
  static_volume:
  frontend_volume:

```

Скриншоты дашборда мониторинга системы:
Состояние сервера(данные node)



Состояние Django(данные django)



Основные View приложения

```
district_cities | questions | models.py | Git: [Icons]
models.py
69 for i in range(len(self.answers)):
70     if self.answers[i][0] == question_id:
71         if self.answers[i][1] == answer:
72             return
73         if self.answers[i][1]:
74             old_answer = self.answers[i][1]
75
76         self.answers[i][1] = answer
77         break
78
79 old_answer_is_correct = question.check_answer(old_answer)
80 new_is_correct = question.check_answer(answer)
81
82 if old_answer_is_correct and not new_is_correct:
83     self.correct_answers_count -= 1
84 elif not old_answer_is_correct and new_is_correct:
85     self.correct_answers_count += 1
86
87 self.save()
88
89 def stop(self) -> bool:
90     for answer in self.answers:
91         if not answer[1]:
92             return False
93
94     self.closed = True
95     self.save()
96     return True
97
98 @staticmethod
99 def get_running_quiz(user: User, city: City):
100     """
101     Function checks is there any stated sessions for user and city
102     """
103     now = datetime.now()
104
105     quiz: List[Quiz] = Quiz.objects.filter(
106         user=user, city=city, time_started__lt=now, closed=False,
107     )
108     for q in quiz:
109         if q.time_started + QUIZ_TIME_MINUTES > now:
110             return q
111
112     return None
113
114 @staticmethod
115 def start_new(user: User, city: City):
116     quiz = Quiz.get_running_quiz(user, city)
117     if quiz is not None:
118         return quiz
119
120     quiz = Quiz(user=user, city=city)
121     questions_ids = Question.get_pks_by_city(city)
122     answers = []
123     for question_id in questions_ids:
124         answers.append([question_id, None])
125     quiz.answers = answers
126     quiz.save()
127     return quiz
128
129 @staticmethod
130 def get_top(city_id=None):
131     if city_id is None:
132         quiz_top: List[Quiz] = Quiz.objects.all().order_by("-correct_answers_count")[
133             :TOP_USERS_COUNT
134         ]
135     else:
136         quiz_top: List[Quiz] = Quiz.objects.filter(city__pk=city_id).order_by(
137             "-correct_answers_count"
138         )[:TOP_USERS_COUNT]
139
140     print(quiz_top, "")
141
142     users = []
143     for quiz_session in quiz_top:
144         users.append([quiz_session.user, quiz_session])
145     return users
146
147 def __str__(self):
148     return f"{self.user}: {self.city}"
149
```



```
district_cities | questions | models.py | district_cities | Git | Database | ScView
models.py
1 import random
2 from datetime import datetime
3 from typing import List
4
5 from django.db import models
6
7 from cities.models import City
8 from district_cities import settings
9 from district_cities.settings import QUIZ_TIME_MINUTES, TOP_USERS_COUNT
10 from users.models import User
11
12
13 class Question(models.Model):
14     city = models.ForeignKey(City, on_delete=models.CASCADE)
15     text = models.TextField()
16     type = models.IntegerField(choices=settings.QUESTION_TYPES)
17     options = models.JSONField()
18     answer = models.CharField(max_length=10)
19
20     @staticmethod
21     def get_pks_by_city(city: City):
22         pks = Question.objects.values_list("pk", flat=True).filter(city=city)
23         random_pks = random.sample(list(pks), settings.QUESTIONS_COUNT_IN_QUIZ)
24         return random_pks
25
26     def check_answer(self, answer):
27         return self.answer == answer
28
29     def __str__(self):
30         return f"{self.city}: {self.text}"
31
32
33 class QuestionImages(models.Model):
34     question = models.ForeignKey(Question, on_delete=models.CASCADE)
35     image = models.ImageField(upload_to="questions/")
36
37     def __str__(self):
38         return str(self.question)
39
40
41 class QuizEndedException(Exception):
42     pass
43
44
45 class Quiz(models.Model):
46     user = models.ForeignKey(User, on_delete=models.PROTECT)
47     city = models.ForeignKey(City, null=True, blank=True, on_delete=models.PROTECT)
48     time_started = models.DateTimeField(auto_now_add=True)
49     answers = models.JSONField()
50     correct_answers_count = models.IntegerField(default=0)
51     closed = models.BooleanField(default=False)
52
53     def is_closed(self):
54         if self.time_started + QUIZ_TIME_MINUTES < datetime.now():
55             return True
56
57         if self.closed:
58             return True
59
60         return False
61
62     def check_answer(self, question_id: int, answer: str):
63         if self.is_closed():
64             raise QuizEndedException
65
66         question = Question.objects.get(pk=question_id)
67
68         old_answer = None
69         for i in range(len(self.answers)):
70             if self.answers[i][0] == question_id:
71                 if self.answers[i][1] == answer:
72                     return
73                 if self.answers[i][1]:
74                     old_answer = self.answers[i][1]
75
76                 self.answers[i][1] = answer
77                 break
78
79         old_answer_is_correct = question.check_answer(old_answer)
80         new_is_correct = question.check_answer(answer)
81
82         if old_answer_is_correct and not new_is_correct:
83             self.correct_answers_count -= 1
84         elif not old_answer_is_correct and new_is_correct:
85             self.correct_answers_count += 1
```

Реализация компонента на примере “Вопроса викторины”

```
import React from "react";
import './Question.scss'
// @ts-ignore

import ...

type QuestionProps = {
  isLast: boolean;
  quizId: number;
  cityId: number;
  question: any;
}

export const Question: React.FC<QuestionProps> = ({ isLast, cityId, quizId, question }: QuestionProps) => {

  const store = useStore();

  const fetchQuestions = React.useCallback( callback: ()=>{...}, deps: []);

  const imageElements = React.useMemo( factory: ():React.ReactNode=>{...}, deps: [question.images])

  const sendAnswer = React.useCallback( callback: ()=>{...}, deps: [quizId, question.id, question.answers])

  const select = React.useCallback( callback: (option:string, optIndex: number)=>{...}, deps: [question.answers,question.options])
  const optionElements = React.useMemo( factory: (): React.ReactNode=>{...}, deps: [question.options, question.answers])

  return (
    <div className="question"...>
  );
}
```

```
const sendAnswer = React.useCallback( callback: ()=>{
  apiProxy(
    endpoint: `${serverUrl}quiz/${quizId}/questions/${question.id}`,
    methodType: 'POST', {...},
    config: undefined, multipartFormData: false, withCsrf: true).then(responseAxios => {...})
  }, deps: [quizId, question.id, question.answers])

const select = React.useCallback( callback: (option:string, optIndex: number)=>{
  const ansIndex = question.answers.indexOf(optIndex);
  if (ansIndex === -1) {
    if(question.type === QuestionType.ONE){...} else if(question.type === QuestionType.MANY){...}
    sendAnswer();
    return
  }
  question.answers.forEach((el: number)=>{
    if(el === optIndex){
      const arr = question.answers.filter((el: any)=>{...})
      store.dispatch(addAnswerToQuestion(question.id, arr))
      sendAnswer();
      return
    }
  })
}, deps: [question.answers,question.options])

const optionElements = React.useMemo( factory: (): React.ReactNode=>{
  return question.options.map((option: {} | null | undefined, optIndex: number)=>{
    const isSelected = question.answers.indexOf(optIndex) !== -1
    return(
      <div className="option-element">
        <img src={isSelected ? indicatorActive : indicator} onClick={()=>select(option as string,optIndex)}>
        <span>{option}</span>
      </div>
    )
  })
}, deps: [question.options, question.answers])
```

Ссылка на проект

<http://russian-town.ru>