

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
УНИВЕРСИТЕТ ИТМО

Факультет инфокоммуникационных технологий

Образовательная программа 09.03.02

Направление подготовки (специальность) Мобильные сетевые технологии

О Т Ч Е Т

о курсовой работе

Тема задания: реализация серверной и визуальной части веб-приложения для создание личностных новостей с возможностью делиться ими с пользователями, средствами Django, Django REST Framework, Vue3

Обучающийся Кумпан Виктор Викторович, К33401

Руководитель: Говоров А. И., преподаватель

Оценка за курсовую работу ____

Подписи членов комиссии:

_____(Говоров А. И.)
(подпись)

_____(Чунаев А. В.)
(подпись)

_____(Антонов М. Б.)
(подпись)

Дата ____

Санкт-Петербург
2022

ВВЕДЕНИЕ	3
Актуальность	3
Цели и задачи	3
ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ И ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ	4
1.1 Средства разработки	4
1.2 Функциональные требования	4
ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ	4
2.1. Проектирование и реализация приложения	4
2.2. Backend часть приложения	5
2.2.1 Реализация API urls	5
2.2.2 Реализация представления	6
2.2.3 Реализация сериализаторов	10
2.2.4 Реализация моделей	13
2.2.5 Сборка backend части	15
2.3. Реализация frontend части	16
2.3.1 Реализация router	16
2.3.2 Реализация страниц	18
2.3.3 Реализация миксинов	27
2.3.4 Скрипкаст веб-приложения	31
ЗАКЛЮЧЕНИЕ	39
СПИСОК ЛИТЕРАТУРЫ	40

ВВЕДЕНИЕ

Актуальность

Мессенджеры становятся сильно перегружены информацией и большинство людей замечают, что начальной целью входа в соцсеть было написание сообщения другу, а в итоге остановился на просмотре видео. Это значительно снижает доверие некоторого сегмента пользователей. Мой сервис позволяет абстрагироваться от внешних источников контента и сфокусироваться на записи своих новостей (возможно целей) и делиться ими с окружающими.

Цели и задачи

1. Определение средств разработки
2. Определение функциональных требований
3. Реализация адресов
4. Реализация представление
5. Реализация сериализаторов
6. Реализация frontend части приложения

ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ И ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1.1 Средства разработки

Для разработки backend части веб-приложения использовались следующие инструменты: Python, Django REST framework, Docker. Выбор данного стека обусловлен тем что, Python (Django R. f.) предоставляет методы для быстрой и достаточно производительной серверной части, также значительно ускоряет процесс разработки. Для frontend части веб-приложения использовался Vue.js framework. Выбор обусловлен тем, что данный фреймворк предоставляет удобное внутреннее API, которое содержит все необходимые инструменты для быстрой и качественной разработки визуала приложения.

1.2 Функциональные требования

1. Компоненты страниц охватывающие следующий функционал
 - a. Логин
 - b. Регистрация
 - c. Сброс пароля
 - d. Сортировка новостей
 - e. Профайл пользователей
 - f. Прочие содержащий опциональный функционал
2. Компоненты backend и frontend части охватывающие следующий функционал:
 - a. Миксины во Vue
 - b. Фильтрация запросов по API в Django REST

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

2.1. Проектирование и реализация приложения

Двумя основными сущностями веб-приложения является пользователь и новости, которые он может создавать и делиться ими. Все действия в приложении доступны только после регистрации и входа пользователя. Пользователь может изменять свои личные данные в личном кабинете, также есть возможность удалять новости, редактировать и детально просматривать их. Пользователь может ознакомиться с профайлом других пользователей.

2.2. Backend часть приложения

2.2.1. Реализация API urls

В ходе разработки разделил API urls на 3 основных блока аутентификация, профайл, заметки (новости). Данный подход позволяет сделать API более интуитивно понятным. В некоторых url использовал именованные аргументы для создания гибкого интерфейса: <int:pk>, <slug>.

```
from django.urls import path, include

from rest_framework.routers import DefaultRouter
from .views import SignupAPIView, NoteViewSet, \
    Public, Logout, ProfileView, ConfidentProfileView, \
    NoteViewDetail, NoteDetailCreate, PublicViewDetail
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView, TokenVerifyView

from rest_framework.authtoken.views import obtain_auth_token


app_name = 'api'

urlpatterns = [
    path('signup/', SignupAPIView.as_view(), name='signup'),
    path('token/', obtain_auth_token, name='token'),
    path('logout/', Logout.as_view(), name='logout'),

    path('profile/<slug>/', ProfileView.as_view({'get': 'list'}), name='profile_info'),
    path('profile/', ConfidentProfileView.as_view({'get': 'list'}), name='self_pr_info'),
    path('profilee/', ConfidentProfileView.as_view({'put': 'update'}), name='profile_update'),

    path('notes/<int:pk>/edit/', NoteViewDetail.as_view({'put': 'update'}), name='reload'),
    path('notes/delete/<int:pk>/', NoteViewSet.as_view({'delete': 'destroy'}), name='del'),
    path('notes/new/', NoteDetailCreate.as_view({'post': 'create'}), name='create'),
    path('notes/', NoteViewSet.as_view({'get': 'list'}), name='note-list'),
    path('notes/public/', Public.as_view({'get': 'list'}), name='shared'),
    path('notes/public/<int:pk>/', PublicViewDetail.as_view({'get': 'list'}), name='shared'),
    path('notes/<int:pk>/', NoteViewDetail.as_view({'get': 'list'}), name='note-detail'),
]
```

2.2.2 Реализация представлений

Спускаясь на более низкий уровень абстракции перейдем к реализации представлений, используемых при обращении к API. Все нижеприведенные представления наследуются от `rest_framework.views -> APIView` и `rest_framework -> viewsets`

- Представление для регистрации пользователя, в нем был переопределен метод `post`, можем заметить, что происходит проверка на корректность ввода пароля.

```
17 class SignupAPIView(APIView):
18     """
19     API view for sign up API
20     """
21     permission_classes = []
22
23     def post(self, request):
24
25         password = request.POST.get('password', None)
26         confirm_password = request.POST.get('confirm_password', None)
27
28         if password == confirm_password:
29
30             serializer = SignupSerializer(data=request.data)
31             # serializer.create(validated_data=request.data)
32             serializer.is_valid(raise_exception=True)
33             serializer.save()
34
35             data = serializer.data
36             response = status.HTTP_201_CREATED
37         else:
38             data = ''
39             raise ValidationError(
40                 {'password_mismatch': 'Password fields didn\'t match.'})
41     return Response(data, status=response)
```

- В представлении `Logout` изменено поведение на `get` запрос, а именно происходит удаление токена доступа у пользователя

```
73 class Logout(APIView):
74
75     def get(self, request, format=None):
76         request.user.auth_token.delete()
77
78     return Response(status=status.HTTP_200_OK)
```

3. В данном представлении изменена фильтрация, она позволяет передавать информацию о конкретном профайле пользователя.

```
43     class ProfileView(viewsets.ModelViewSet):  
44  
45         serializer_class = ProfileSerializer  
46         queryset = UserProfile.objects.all()  
47         permission_classes = [permissions.IsAuthenticated]  
48         renderer_classes = [JSONRenderer]  
49  
50         def filter_queryset(self, queryset):  
51             name = self.request.parser_context["kwargs"]  
52             queryset = UserProfile.objects.filter(username=name["slug"])  
53             return queryset
```

4. В данном представлении передается информация по пользователю совершившему запрос, также переопределен метод `def get_object(self):` в связи с тем, что использование <slug> в url требует переопределение этого метода в представлении. Также переопределен метод `update`.

```
55     class ConfidentProfileView(viewsets.ModelViewSet):  
56  
57         serializer_class = ConfidentProfileSerializer  
58         queryset = UserProfile.objects.all()  
59         permission_classes = [permissions.IsAuthenticated]  
60         renderer_classes = [JSONRenderer]  
61  
62         def filter_queryset(self, queryset):  
63             return UserProfile.objects.filter(email=self.request.user)  
64  
65         def get_object(self):  
66             return UserProfile.objects.get(email=self.request.user)  
67  
68         def perform_update(self, serializer):  
69             serializer.save()
```

5. Представление для детального отображения заметок(новостей) фильтрация выполняется по уникальному номеру новости полученному по API, также переопределен метод `def perform_update(self, serializer):`

для детального обновление новостей

```
80     class NoteViewDetail(viewsets.ModelViewSet):
81         serializer_class = NoteSerializerDetail
82         queryset = Note.objects.all()
83         permission_classes = [permissions.IsAuthenticated]
84         renderer_classes = [JSONRenderer]
85
86         def filter_queryset(self, queryset):
87             params = self.request.parser_context.get("kwargs")
88             return Note.objects.filter(owner=self.request.user, id=params.get("pk"))
89
90         def perform_update(self, serializer):
91             serializer.save()
```

6. Представление для получения личных заметок пользователя запрос должен содержать аргументы `_page` и `_limit` для удобной реализации пагинации на frontend

```
105    class NoteViewSet(viewsets.ModelViewSet):
106
107        serializer_class = NoteSerializer
108        queryset = Note.objects.all()
109        permission_classes = [permissions.IsAuthenticated]
110        renderer_classes = [JSONRenderer]
111
112        def filter_queryset(self, queryset):
113            page = self.request.query_params.get("_page")
114            limit = self.request.query_params.get("_limit")
115            if page and limit:
116                page, limit = int(page), int(limit)
117                queryset = Note.objects.filter(owner=self.request.user)
118                return queryset[(page-1)*limit:page*limit]
119            queryset = Note.objects.filter(owner=self.request.user)
120            return queryset
121
122        def perform_destroy(self, instance):
123            instance.delete()
```

7. Представление для создания детальных новостей вывел это в отдельное представление так как содержит свою логику и serializer

```
93 class NoteDetailCreate(viewsets.ModelViewSet):  
94     serializer_class = NoteSerializerDetailCreate  
95     queryset = Note.objects.all()  
96     permission_classes = [permissions.IsAuthenticated]  
97     renderer_classes = [JSONRenderer]  
98  
99  
100    def perform_create(self, serializer):  
101        serializer.save(owner=self.request.user)
```

8. Представление для публичных новостей, переопределена фильтрация как и в пункте 6

```
128 class Public(viewsets.ModelViewSet):  
129     serializer_class = NotePublicSerializer  
130     queryset = Note.objects.all()  
131     permission_classes = [permissions.IsAuthenticated]  
132     renderer_classes = [JSONRenderer]  
133  
134     def filter_queryset(self, queryset):  
135         page = self.request.query_params.get("_page")  
136         limit = self.request.query_params.get("_limit")  
137         if page and limit:  
138             page, limit = int(page), int(limit)  
139             queryset = Note.objects.filter(public__icontains=True)  
140             return queryset[(page - 1) * limit:page * limit]  
141         return Note.objects.filter(public__icontains=True)
```

9. Представление для детального просмотра публичных новостей по их индикатору

```
145 class PublicViewDetail(viewsets.ModelViewSet):  
146     serializer_class = NotePublicSerializer  
147     queryset = Note.objects.all()  
148     permission_classes = [permissions.IsAuthenticated]  
149     renderer_classes = [JSONRenderer]  
150  
151     def filter_queryset(self, queryset):  
152         params = self.request.parser_context.get("kwargs")  
153         return Note.objects.filter(public__icontains=True, id=params.get("pk"))
```

10. Представление для регистрации через электронную почту (не используется в работе)

```
156     class Register(viewsets.ModelViewSet):  
157  
158         model = get_user_model()  
159         serializer_class = RegisterSerializer  
160         permission_classes = [permissions.AllowAny]  
161  
162         def perform_create(self, serializer):  
163             serializer.save()
```

2.2.3 Реализация сериализаторов

Реализация сериализаторов основана на применение полиморфизма к классу

`serializers.ModelSerializer` из библиотеки `rest_framework`

1. Сериализатор для регистрации, был переопределен метод `create` с проверкой пароля и созданием пользователя в системе

```
16     class SignupSerializer(serializers.ModelSerializer):  
17  
18         class Meta:  
19             model = UserProfile  
20             fields = ['id', 'username', 'email', 'password']  
21             extra_kwargs = {'password': {'write_only': True}}  
22  
23         def create(self, validated_data):  
24             if validate_password(validated_data['password']) == None:  
25                 password = make_password(validated_data['password'])  
26                 user = UserProfile.objects.create(  
27                     username=validated_data['username'],  
28                     email=validated_data['email'],  
29                     password=password  
30                 )  
31             return user
```

2. Сериалайзер для профайла доступного к просмотру другим пользователям, обратим внимание на поля, отсутствуют пароль и почта.

Сделано для большей конфиденциальности пользователя.

```
34     class ProfileSerializer(serializers.ModelSerializer):
35         class Meta:
36             model = UserProfile
37             fields = ['username', "age", "education",
38                       "first_name", "second_name", "town"]
```

3. Сериалайзер для личного профайла, исключены дефолтные обязательный поля, так как пользователь при обновлении своего профайла не всегда хочет менять пароль и прочие параметры. Для более читабельной и компактной реализации использованы методы `setattr` `getattr` `hasattr`

```
41     class ConfidentProfileSerializer(serializers.ModelSerializer):
42         class Meta:
43             model = UserProfile
44             fields = ['username', 'email', 'password',
45                       "first_name", "second_name", "town",
46                       "age", "education",
47                       ]
48             extra_kwargs = {'username': {'required': False},
49                            'email': {'required': False},
50                            'password': {'required': False},
51                            }
52             # confirm_password = serializers.CharField(required=False)
53
54         def update(self, instance, validated_data):
55             for inst in validated_data:
56                 if inst == "password":
57                     instance.set_password(validated_data["password"])
58                     continue
59                 if hasattr(instance, inst):
60                     setattr(instance, inst, validated_data.get(inst, getattr(instance, inst)))
61             instance.save()
62             return instance
```

4. Сериалайзер для новостей

```
64     class NoteSerializer(serializers.ModelSerializer):
65         class Meta:
66             model = Note
67             fields = ('id', 'title', 'body', 'tags')
68             extra_kwargs = {'title': {'required': False},
69                            'body': {'required': False},
70                            'pub_date': {'required': False},
71                            }
```

5. Сериалайзер для новостей в паблике, переопределил owner потому что по дефолту возвращается id пользователя, а для логики хотелось бы имя

пользователя создавшего новость.

```
74     class NotePublicSerializer(serializers.ModelSerializer):
75         owner = serializers.CharField(source='owner.username')
76
77         class Meta:
78             model = Note
79             fields = ('id', 'owner', 'title', 'body', 'tags', 'pub_date')
80             extra_kwargs = {'title': {'required': False},
81                            'body': {'required': False},
82                            'pub_date': {'required': False},
83                            }
```

6. Сериалайзер для детальной информации по новости, в методе create вызываю функцию у родителей в порядке MRO

```
88     class NoteSerializerDetail(serializers.ModelSerializer):
89         class Meta:
90             model = Note
91             fields = ('id', 'title', 'body', 'tags',
92                       "pub_date", "tags", "public")
93             extra_kwargs = {'title': {'required': False},
94                            'body': {'required': False},
95                            'pub_date': {'required': False},
96                            'tags': {'required': False},
97                            }
98
99         def create(self, validated_data):
100            instance = super().create(validated_data)
101            return instance
102
103        def update(self, instance, validated_data):
104            for inst in validated_data:
105                if hasattr(instance, inst):
106                    setattr(instance, inst, validated_data.get(inst, getattr(instance, inst)))
107            instance.save()
108            return instance
```

7. Сериалайзер для регистрации пользователя с подтверждением почты

```
125 class RegisterSerializer(serializers.ModelSerializer):
126     password = serializers.CharField(write_only=True)
127
128     def create(self, validated_data):
129
130         user = UserProfile.objects.create(
131             email=validated_data['email']
132         )
133         user.set_password(validated_data['password'])
134         user.is_active = False
135
136         # current_site = get_current_site(self)
137         mail_subject = 'Activate your blog account.'
138         message = render_to_string('acc_active_email.html', {
139             'user': user,
140             'domain': "127.0.0.1:8000",
141             'uid': urlsafe_base64_encode(force_bytes(user.pk)).decode('utf-8'),
142             'token': account_activation_token.make_token(user),
143         })
144
145         to_email = user.email
146         email = EmailMessage(
147             mail_subject, message, to=[to_email]
148         )
149         email.send()
150         user.save()
151
152         return user
153
154     class Meta:
155         model = UserProfile
156         fields = ('id', 'email', 'password')
157
```

2.2.4 Реализация моделей

Веб приложение имеет две сущности пользователь и новости. Связь один ко многим через id пользователя к заметки.

- Пользователь. Выполнено наследование от `AbstractBaseUser` для создания кастомного интерфеса пользователя уникальными полями является почта и его nickname. Объект мы берем с UserManager, который имеет стандартные методы.

```
35     class UserProfile(AbstractBaseUser):
36         email = models.EmailField(verbose_name='email address',
37                                 max_length=255, unique=True,
38                                 )
39         username = models.CharField(_('username'),
40                                     max_length=150,
41                                     unique=True,
42                                     error_messages={
43                                         'unique': _("A user with that username already exists."),
44                                     },
45                                     )
46         is_active = models.BooleanField(default=True)
47         is_staff = models.BooleanField(default=False)
48         is_admin = models.BooleanField(default=False)
49         USERNAME_FIELD = 'username'
50         REQUIRED_FIELDS = ['email', ]
51
52         first_name = models.CharField(max_length=25, default='')
53         second_name = models.CharField(max_length=25, default='')
54         town = models.CharField(max_length=25, default='')
55         age = models.CharField(max_length=3, default='')
56         education = models.CharField(max_length=30, default='')
57
58         objects = UserManager()
59
60     def __str__(self):
61         return self.email
62
63     def get_full_name(self):
64         return self.email
65
```

```
62
63         def get_full_name(self):
64             return self.email
65
66         def get_short_name(self):
67             return self.email
68
69         def has_perm(self, perm, obj=None):
70             return True
71
72         def has_module_perms(self, app_label):
73             return True
```

```
8 class UserManager(BaseUserManager):
9     use_in_migrations = True
10
11     def create_user(self, email, username, password=None):
12         if email is None:
13             raise ValueError('User must have an email address.')
14         user = self.model(email=self.normalize_email(email), username=username)
15         user.set_password(password)
16         user.save(using=self._db)
17         return user
18
19     def create_superuser(self, email, username, password=None):
20
21         if password is None:
22             raise ValueError('Superusers must have a password.')
23         user = self.create_user(email, username=username, password=password)
24         user.is_superuser = True
25         user.is_staff = True
26         user.save(using=self._db)
27         return user
```

2. Новость

```
71 class Note(models.Model):
72     title = models.CharField(max_length=200)
73     body = models.TextField()
74     pub_date = models.DateTimeField('date published', auto_now_add=True)
75     owner = models.ForeignKey(UserProfile, related_name='notes', on_delete=models.CASCADE)
76     tags = models.CharField(max_length=200)
77     public = models.BooleanField(default=False)
78
79     def was_published_recently(self):
80         now = timezone.now()
81         return now - timezone.timedelta(days=1) <= self.pub_date <= now
82
83     def __str__(self):
84         return self.title
85
```

2.2.5 Сборка backend части

Развертка backend части осуществляется в docker контейнерах ниже приведу docker файлы.

```

1 ➤  FROM python:3.7.7
2
3   ENV PYTHONUNBUFFERED 1
4   RUN mkdir /config
5   COPY requirements /config/requirements
6   RUN pip install -r /config/requirements/dev.txt
7   RUN mkdir /src;
7   WORKDIR /src
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
1  version: '3'
2 ➤   services:
3   ➤     nginx:
4     image: nginx:latest
5     container_name: ng01
6     ports:
7       - "8000:8000"
8     volumes:
9       - ./deploy/nginx:/etc/nginx/conf.d
10      - ./static:/static
11     depends_on:
12       - web
13   ➤     web:
14     build: .
15     container_name: dg01
16     command: bash -c "sleep 5 && python3 manage.py makemigrations && python3 manage.py migrate && uwsgi --ini
17     depends_on:
18       - db
19     volumes:
20       - ./src:/src
21       - ./deploy/uwsgi:/usr/local/etc/
22       - ./static:/static
23     expose:
24       - "8000"
25   ➤     db:
26     image: postgres:latest
27     container_name: ps01
28     environment:
29       POSTGRES_DB: "db"
30       POSTGRES_HOST_AUTH_METHOD: "trust"

```

2.3. Frontend часть

Основной идеей в реализации данной части было грамотное применение в наследовании компонентов, использование mixins, верстка собственных UI объектов, связь интерфеса с backend с использованием Axios.

2.3.1 Реализация router

Основные роутеры в ве-приложении привожу ниже, некоторые из них с динамически изменяемы :id

```
1  import SignUp from "@/pages/SignUp";
2  import SignIn from "@/pages/SignIn";
3  import {createRouter, createWebHistory} from "vue-router";
4  import NewsMy from "@/pages/NewsMy";
5  import NewsPublic from "@/pages/NewsPublic";
6  import Profile from "@/pages/Profile";
7  import NewsDetail from "@/pages/NewsDetail";
8
9  const routes = [
10    {
11      path: '/news',
12      component: NewsMy
13    },
14    {
15      path: '/news/public',
16      component: NewsPublic
17    },
18    {
19      path: '/news/public/:id/',
20      component: NewsDetail
21    },
22    {
23      path: '/news/:id/',
24      component: NewsDetail
25    },
26    {
27      path: '/auth/signup',
28      component: SignUp,
29    },
30    {
31      path: '/auth/signin',
32      component: SignIn
33    }
]
```

```

35           path: '/profile',
36           component: Profile
37     },
38   {
39     path: '/profile/:id/',
40     component: Profile
41   },
42 ]
43 const router = createRouter( options: {
44   routes,
45   history: createWebHistory(process.env.BASE_URL)
46 })
47
48 export default router;

```

2.3.2 Реализация страниц

1. Главный компонент приложения App.vue в нем определил навигационную

The screenshot shows the code for the App.vue component. It includes a template section with a navbar and a router-view, a script section importing the Navbar component and defining a components object, and a style section.

```

<template>
  <navbar></navbar>
  <div>
    <router-view></router-view>
  </div>
</template>

<script>
  import Navbar from "@/components/Navbar";
  ...
  export default {
    components: {
      Navbar,
    }
  }
</script>

<style>
</style>

```

панель и вложенный маршрут

2. Страница собственных новостей. Техники используемые на данной странице:

```

<template>
  <div>
    <my-button class="create__btn" @click="showDialog">
      Create post
    </my-button>

    <my-select class="select__man"
      v-model="selectedSort"
      :options="sortOption"
    />

    <my-dialog v-model:show="dialogVisible">
      <post-from @create="CreatePost"
      />
    </my-dialog>
    <post-list
      :posts="sortedNews"
      :loading="loading"
      @remove="removePost"
    >
    </post-list>
    <div ref="observer" class="observer"></div>
  </div>
</template>

```

- a. Пагинации при слежке за объектом. Применение хука.

```

<div
  ref="observer" class="observer"></div>
  async getNews () {
    try {
      console.log(this.page)
      let new_posts = await this.newsRequest({ url: 'api/notes', data: {}, method: "GET", params: {} })
        _page: this.page,
        _limit:this.limit
    });
    this.loading = false
    this.posts = [...this.posts, ...new_posts]
    this.page+=1
  }
  mounted() {
    this.getNews();
    const options = {
      rootMargin: '0px',
      threshold: 1.0
    }
    const callback = (entries, observer) => {
      if ((entries[0].isIntersecting) && (this.page <= this.totalPage)) {
        this.getNews()
      }
    };
    const observer = new IntersectionObserver(callback, options);
    //передаем аргумент за которым следим
    observer.observe(this.$refs.observer)
  }
}

```

- b. Передача `:posts="sortedNews"` параметров (данных) в дочерние компоненты через computed сортирующее свойство.

```
computed: {
  sortedNews(){
    return [...this.posts].sort( compareFn: (post1, post2,)=>post1[this.selectedSort]? .localeCompare(post2[this.selectedSort]))}
},
```

- c. Удаление поста

```
removePost(post){
  const response = this.newsRequest( url: `api/notes/delete/${post.id}`, data: {}, method: "DELETE", params: {})
  this.posts = this.posts.filter(t => t.id !== post.id)
},
```

- d. Создание поста

```
CreatePost(event){
  const response = this.newsRequest( url: 'api/notes/new', event, method: "POST")
  this.posts.push(event)
  this.dialogVisible = false
},
```

- e. Прямая связь select с моделью через v-model

```
<my-select class="select man" v-model="selectedSort">
```

- f. Связь с диалоговым окном для задания соответствующей логики

```
<my-dialog v-model:show="dialogVisible">
```

- g. Определение метода на слушание события

```
<my-button class="create__btn" @click="showDialog">
  Create post
</my-button>
```

- h. Проброс через emit с дочерних классов в родительский

```
@remove="removePost"
```

3. Детальная страница поста

```
<template>
  <post-detail-item :post="post"
    :key="post.id"
    :public="isPublic"
    @remove="removePost"
    @update="updatePost"
  />
</template>
```

Техники

используемые на данной странице:

- Методы удаления, получения данной новости (определен в зависимости от статуса новости (в паблике или локально)), изменение новости

```

updatePost(post){
  if (post.public === ''){
    post.public = false
  }
  this.post = newsRequest.methods.reload_data(post, this.post)
  const response = this.newsRequest({ url: `api/notes/${this.$route.params.id}/edit`, post, method: "PUT"})
}

removePost() {
  const response = this.newsRequest({ url: `api/notes/delete/${this.$route.params.id}`, data: {}, method: "DELETE"})
  this.$router.push('/news/')
},

async getNews (publ) {
  try {
    let data;
    if (publ){
      data = await this.newsRequest({ url: `api/notes/public/${this.$route.params.id}`}, data: {}, method: "GET");
    }else{
      data = await this.newsRequest({ url: `api/notes/${this.$route.params.id}`}, data: {}, method: "GET");
    }
    this.post = data[0]
  } catch (error) {
    console.error('AN API ERROR:', error)
  }
},

```

- b. Computed свойство для определения нахождении заметки (от этого зависит доступность методов на редактирование)

```

computed:{
  isPublic(){
    return this.$route.fullPath.includes("public")
  }
},

```

- c. Mounted хук для подгрузки новости

```

mounted() {
  this.getNews(this.isPublic)
}

```

4. Публичная страница. Ее техническая часть практически полностью совпадает с личной, можно было вынести компонент как страница новостей и определять логику через переменную паблика, тем самым получилось сжать эти

компоненты в одну.

```
NewsPublic.vue
```

```
1 <template>
2   <div class="app__bt�s">
3     <my-select
4       v-model="selectedSort"
5       :options="sortOption"
6     /></div>
7     <post-list
8       :posts="sortedNews"
9       :public="public"
10      :loading="loading"
11    />
12    <div ref="observer" class="observer"></div>
13  </template>
14  <script>
```

5. Страница с профайлом

```
<template>
  <profile-item :prof="prof"
    :selfprofile="this.isSelfProfile"
    @dialog="showDialog">
    </profile-item>

    <div v-if="this.isSelfProfile">
      <my-dialog v-model:show="dialogVisible">
        <profile-form @create="updateProfile">
        />
      </my-dialog>
    </div>
</template>
```

Техники используемые на данной странице:

- Получение сведений о профайле в зависимости от статуса пользователя (он является владельцем или просматривает чей-то профайл) это реализовано через миксину (о них ниже).

```
async getPfoile() {
  let data;
  if (this.isSelfProfile) {
    console.log("GET MY PROFILE")
    data = await this.newsRequest( url: 'api/profile', data: {}, method: "GET")
  } else {
    console.log("GET OUR PROFILE")
    data = await this.newsRequest( url: `api/profile/${this.$route.params.id}`, data: {}, method: "GET")
  }
  this.prof = data[0]
},
```

- Обновление профайла

```
updateProfile(event) {
  this.prof = newsRequest.methods.reload_data(event, this.prof)
  const response = this.newsRequest( url: 'api/profileee', event, method: "PUT")
  this.dialogVisible = false
}
```

6. Страница входа

```
1 <template>
2   <base-auth-layout>
3     <sign-in-form />
4   </base-auth-layout>
5 </template>
6 <script>
7   import SignInForm from '@/components/auth/SignInForm.vue'
8   import BaseAuthLayout from '@/layouts/BaseAuthLayout.vue'
9   export default {
10     name: "SignIn",
11     components: { SignInForm, BaseAuthLayout }
12   }
13 </script>
```



```
<template>
  <form @submit.prevent="login">
    <div class="form-group">
      <label for="username">Login:</label>
      <input v-model="form.username" type="text" id="username" placeholder="Login..."/>
    </div>
    <div class="form-group">
      <label for="password">Password:</label>
      <input v-model="form.password" type="password" id="password" placeholder="Password..."/>
    </div>
    <button variant="primary" type="submit">Login</button>
    <p class="mt-3">Not registered yet? <a href="/auth/signup">Registered</a></p>
  </form>
</template>
<script>
```

Техники используемые на данной странице:

- a. Изменение в поведении страницы (убираем обновление при подтверждении формы) `<form @submit.prevent="login">`
- b. Связь данных формы с моделью `v-model="form.username"`
- c. Реализация редиректа на текст `Registered`

d. Методы на авторизацию

```
30      mixins: [ authRequest ],
31      methods: {
32        login: async function () {
33          try {
34            console.log(this.form)
35            const response = await this.authRequest({ url: 'api/token', this.form })
36            this.setLogined(response.token)
37            await this.$router.push('/news')
38          } catch (error) {
39            console.error('AN API ERROR:', error)
40          }
41        },
42        setLogined (token) {
43          console.log(token);
44          localStorage.setItem('token', token);
45        }
46      }
}
```

7. Страница регистрации

```
1  <template>
2    <form @submit.prevent="register">
3      <h2>Sign Up</h2>
4      <fieldset>
5        <div class="form-group required">
6          <label class="control-label" >Email:</label>
7          <input v-model="form.email" type="text" id="email" placeholder="Email..." />
8          <p><small class="text-muted">The minimum length of a email is 5 characters.</small></p>
9        </div>
10       <div class="form-group required">
11         <label class="control-label" >Nickname:</label>
12         <input v-model="form.username" type="text" id="username" placeholder="Nickname..." />
13         <p><small class="text-muted">The minimum length of a nickname is 5 characters.</small></p>
14       </div>
15       <div class="form-group required">
16         <label class="control-label" >Password:</label>
17         <input v-model="form.password" type="password" id="password" placeholder="Password..." />
18         <p><small class="text-muted">The minimum length of a password is 6 characters.</small></p>
19       </div>
20
21       <div class="form-group required">
22         <label class="control-label" >Repeat password:</label>
23         <my-input
24           v-model="form.confirm_password"
25           type="password"
26           id="repeatPassword"
27           placeholder="Repeat password..." />
28       </div>
29     </div>
30   </fieldset>
31
32
```

```

32      <p class="text-danger" v-if="!form.password.minLength">Длина пароля меньше 6 символов</p>
33      <p
34          class="text-danger"
35          v-if="isPasswordTheSame"
36      >
37          Введённые пароли не совпадают
38      </p>
39      <button variant="primary" type="submit">Регистрация</button>
40      <p class="mt-2">
41          <small class="text-muted">
42              Все поля отмеченные <span class="text-danger">*</span> обязательны для заполнения.
43          </small>
44      </p>
45      <p class="mt-3">Уже есть аккаунт? <a href="/auth/signin">Вход</a></p>
46  </form>
47</template>
48<script>

```

Техники используемые на данной странице:

```

69      validations: {
70          form: {
71              username: {
72                  required,
73                  minLength: minLength( length: 5)
74              },
75              password: {
76                  required,
77                  minLength: minLength( length: 6)
78              },
79              confirm_password: {
80                  required,
81                  sameAs: sameAs( field: 'password')
82              }
83          }
84      },

```

- a. Валидация форм
- b. computed свойства на корректность формы

```

86      isPasswordTheSame () {
87          const form = this.form
88          return form.password.required
89              && form.confirm_password.required
90              && !form.confirm_password.sameAs
91      }

```

c. Логика на регистрацию

```
94     mixins: [ authRequest ],
95
96     methods: {
97       async register () {
98         try {
99           await this.authRequest({ url: 'api/signup', this.form })
100          await this.$router.push('/auth/signin')
101        } catch (e) {
102          console.error('AN API ERROR', e)
103          this.err = e
104        }
105      },
106    }
107  }
108
```

2.3.3 Реализация миксинов

В ходе разработки и проектирования интерфеса были проанализированы части, которые обладают схожей логикой и дублирование их приведет к раздуванию кода и к сложности быстрого изменения функционала приложения. Ниже я приведу основные миксины.

1. Аутентификация

```
1  const authRequest = {
2    methods: {
3      async authRequest (url, data) {
4        const DEFAULT_HEADERS = { 'Content-type': 'application/json' }
5        const BASE_URL = 'http://localhost:8000'
6        const __url = `${BASE_URL}/${url}/`
7
8        const response = await this.axios({ config: {
9          method: 'POST',
10         url: __url,
11         data,
12         headers: DEFAULT_HEADERS
13       })
14
15       if (response.status !== 200 && response.status !== 201) {
16         throw new Error(response.error)
17       }
18       return response.data
19     }
20   }
21 }
22
23 export default authRequest
```

2. Булевые миксины для проверки аутентификации и статуса профайла, подаются напрямую в шаблон или в дочерние компоненты

```
isAuth.js ×
1 import {len} from "vuelidate/lib/validators/common";
2
3 const isAuth = {
4   computed: {
5     isAuthenticated(){
6       return localStorage.getItem("key: "token") != null
7     },
8     isSelfProfile(){
9       return len(this.$route.params) === 0;
10    },
11  },
12}
13 export default isAuth
```

3. Отправка запросов по API в зависимости от метода запроса тела запроса и передаваемых параметров. Самый используемый миксин в логике проекта

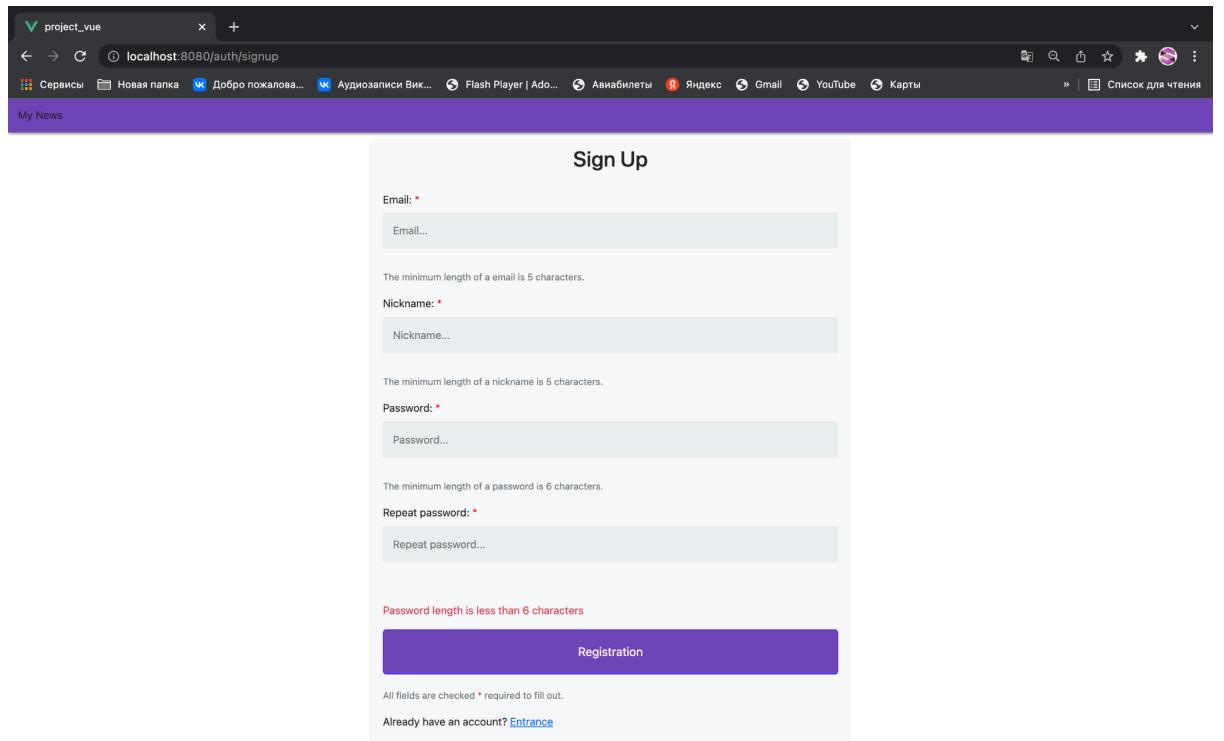
```
1 import {len} from "vuelidate/lib/validators/common";
2
3 const newsRequest = {
4   methods: {
5     check_data(data) {...},
6     reload_data(new_data, curr_data){...},
7     async newsRequest(url, data:{} ={}, method, params:{} = {}) {
8       const token = "Token"+' '+localStorage.token
9       const DEFAULT_HEADERS = { 'Authorization':token}
10      const BASE_URL = 'http://localhost:8000'
11      const __url = `${BASE_URL}/${url}/`
12
13      if (len(data) > 0) {
14        data = this.check_data(data)
15      }
16
17      const response = await this.axios( config: {
18        method: method,
19        url: __url,
20        data: data,
21        params: params,
22        headers: DEFAULT_HEADERS
23      })
24
25      if (response.status !== 200 && response.status !== 201) {
26        throw new Error(response.error)
27      }
28
29      return response.data
30    }
31  }
32}
33
34
35
36
37
38
39
40
41
42
43
44
45 export default newsRequest
```

4. Миксин для работы с диалоговым окном.

```
export default {
  props: {
    show: {
      type: Boolean,
      required: true,
    }
  },
  methods: {
    hideDialog() {
      this.$emit('update:show', false)
    }
  }
}
```

2.3.4 Скринкаст веб-приложения

1. Регистрация



The screenshot shows a registration form titled "Sign Up" on a purple-themed website. The form includes fields for Email, Nickname, Password, and Repeat password. Error messages are displayed for the password field: "The minimum length of a password is 6 characters." and "Password length is less than 6 characters". A "Registration" button is present, along with links for "All fields are checked * required to fill out." and "Already have an account? [Entrance](#)".

Email: *

Nickname: *

Password: *

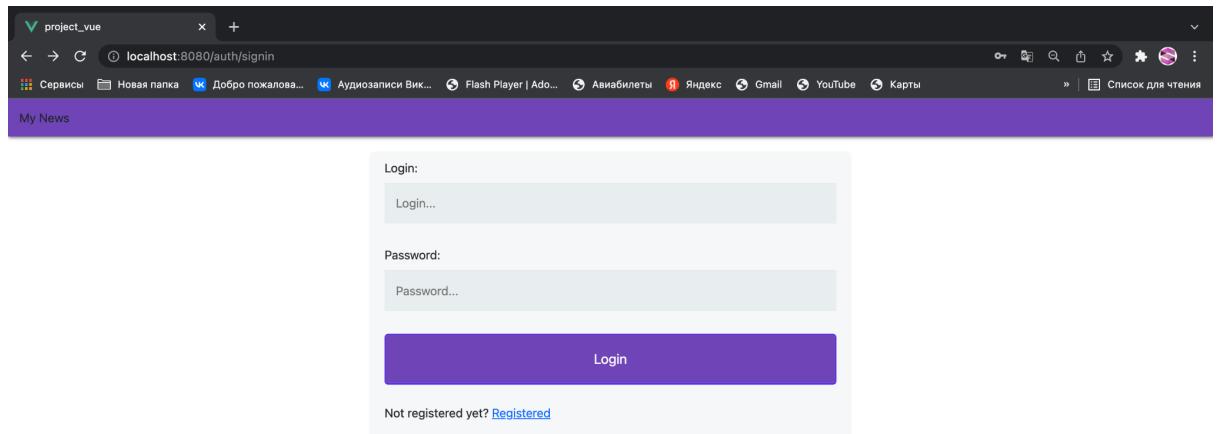
Repeat password: *

Registration

All fields are checked * required to fill out.

Already have an account? [Entrance](#)

2. Авторизация



The screenshot shows a login form on the same purple-themed website. It features fields for Login and Password, and a "Login" button. Below the button, there is a link for users who have not registered yet: "Not registered yet? [Registered](#)".

Login:

Password:

Login

Not registered yet? [Registered](#)

3. Личные новости

The screenshot shows a web application interface with a purple header bar. The header contains the following elements from left to right: 'My News' (with a green triangle icon), 'Love Topics' (with a blue circle icon), 'Go public' (button), 'Profile' (button), and 'Logout' (button). Below the header is a dropdown menu labeled 'Select from the list'. A message 'Post list is empty' is displayed in blue text.

Below the header, the main content area has a title 'Self posts' in bold black text. Underneath it, there is a list of five items, each enclosed in a light blue box:

- 1**
Title:1231
Body:2312312
Tags:3123123
Buttons: Open note, Delete note
- 2**
Title:123123
Body:12312
Tags:31231233
Buttons: Open note, Delete note
- 3**
Title:12312
Body:312312312
Tags:123123
Buttons: Open note, Delete note
- 4**
Title:123123
Body:123123
Tags:12312312
Buttons: Open note, Delete note
- 5**
Title:3123
Body:21312321
Tags:12312
Buttons: Open note, Delete note

The screenshot shows a web application interface for managing posts. At the top, there is a navigation bar with links like 'My News', 'Love Topics', 'Go public', 'Profile', and 'Logout'. A dropdown menu labeled 'Select from the list' is also present.

The main area is titled 'Self posts' and displays a list of five posts, each with a title, body, and tags. To the right of each post are 'Open note' and 'Delete note' buttons. A 'Create post' button is located at the bottom of the list.

A modal window titled 'Create post' is open in the center. It contains input fields for 'Title...', 'Description...', and 'Tag...'. There is also a checkbox for 'Public status' and a 'Create!' button.

Below the first list, there is a sorting dropdown set to 'Sort by body'. Another 'Create post' button is located below this dropdown.

The second section of the screenshot shows the same 'Self posts' list, but the sorting dropdown is now set to 'Sort by title'. The posts are listed in ascending order of title.

The third section shows the same list again, but the sorting dropdown is set to 'Sort by body' again, demonstrating the ability to switch between different sorting criteria.

4. Детальное представление новости

Title
1231

Body
2312312

Tags
3123123

Public status
Private

Date create
2022-01-24T18:46:40.762867+03:00

Update **Delete note**

Title
1231

Body
2312312

Tags
3123123

Public status
Private

Date create
2022-01-24T18:46:40.762867+03:00

Create post
Title...
Description...
Tag...
 Public status
Create!

Update **Delete note**

5. Паблик

The screenshot shows a web browser window with the URL `localhost:8080/news/public`. The page has a purple header bar with buttons for "My News", "Love Topics", "Go public", "Profile", and "Logout". Below the header is a section titled "Public posts" containing four items, each with a small "Open note" button:

- 1
Owner:skyyy
Title:Hello
Body:world
Tags:OK
- 2
Owner:skyyy
Title:Normaly
Body:Ok
Tags:
- 3
Owner:skylar
Title:fsd
Body:dsf
Tags:sdf
- 4
Owner:skylar
Title:sdf
Body:dsf
Tags:sdf

A dropdown menu in the top right corner says "Select from the list", "Sorted by title", and "Sorted by body".

The screenshot shows a web browser window with the URL `localhost:8080/news/public/104`. The page has a purple header bar with buttons for "My News", "Love Topics", "Go public", "Profile", and "Logout". Below the header is a detailed view of a post:

Owner:
skyyy

Title
Hello

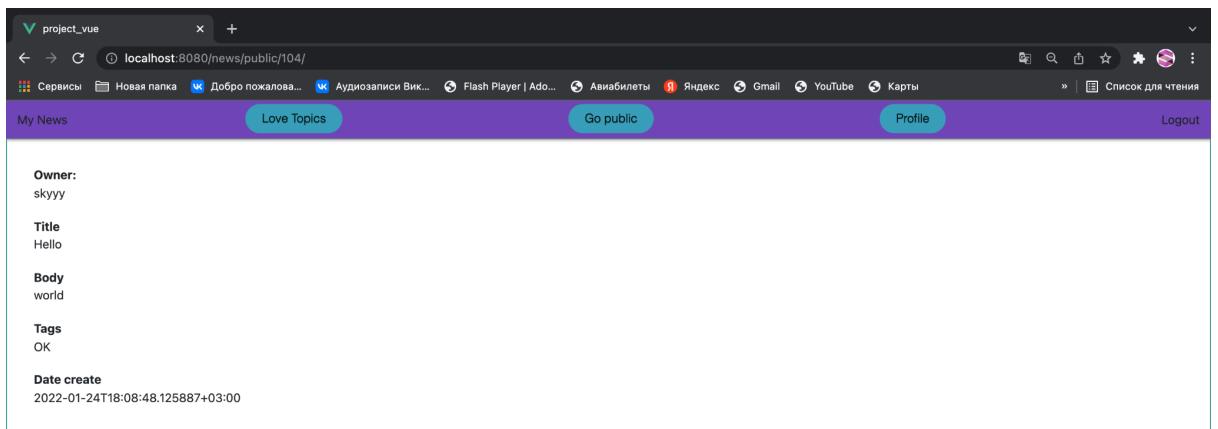
Body
world

Tags
OK

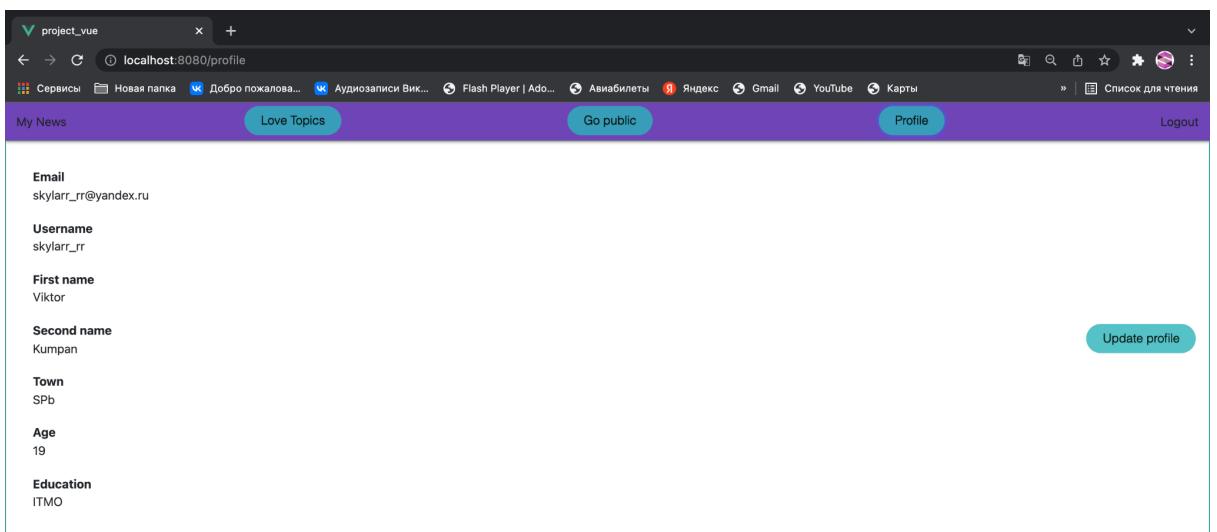
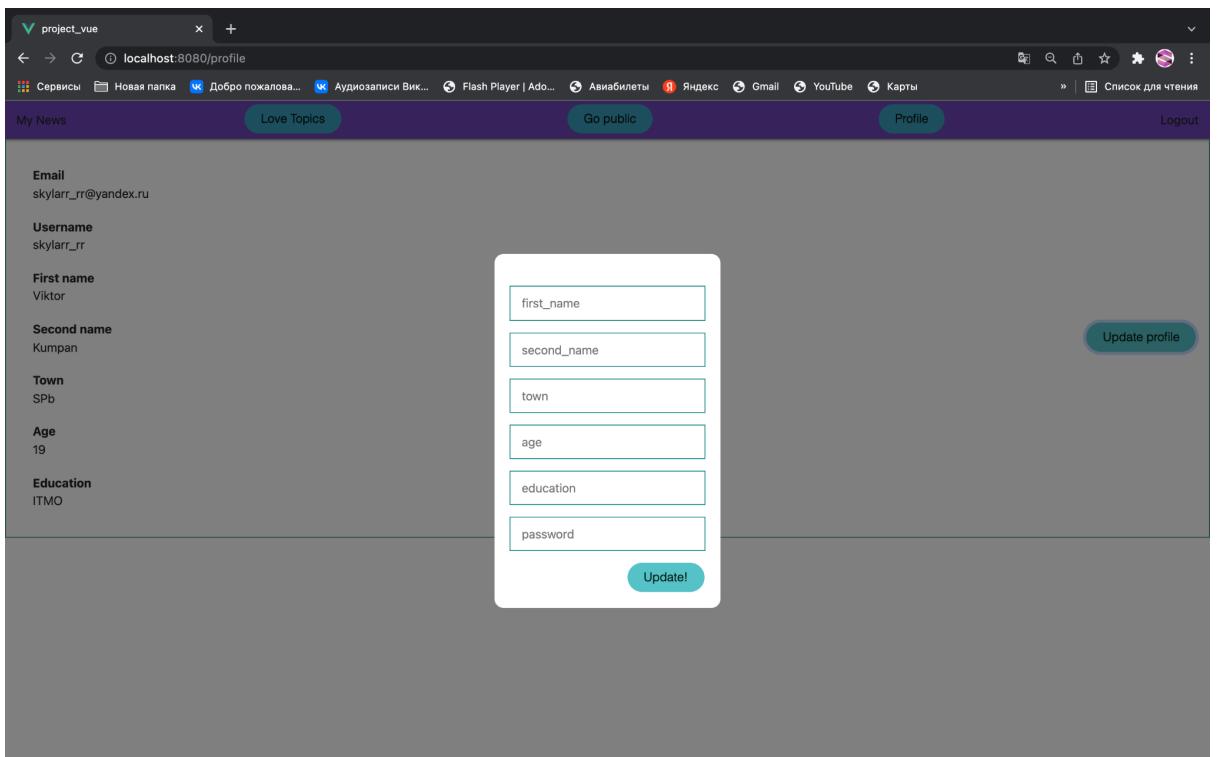
Date create
2022-01-24T18:08:48.125887+03:00

The screenshot shows a web browser window with the URL `localhost:8080/news/public`. The page has a purple header bar with buttons for "My News", "Love Topics", "Go public", "Profile", and "Logout". A dropdown menu in the top right corner says "Sorted by title". Below the header is a section titled "Public posts" containing four items, each with a small "Open note" button:

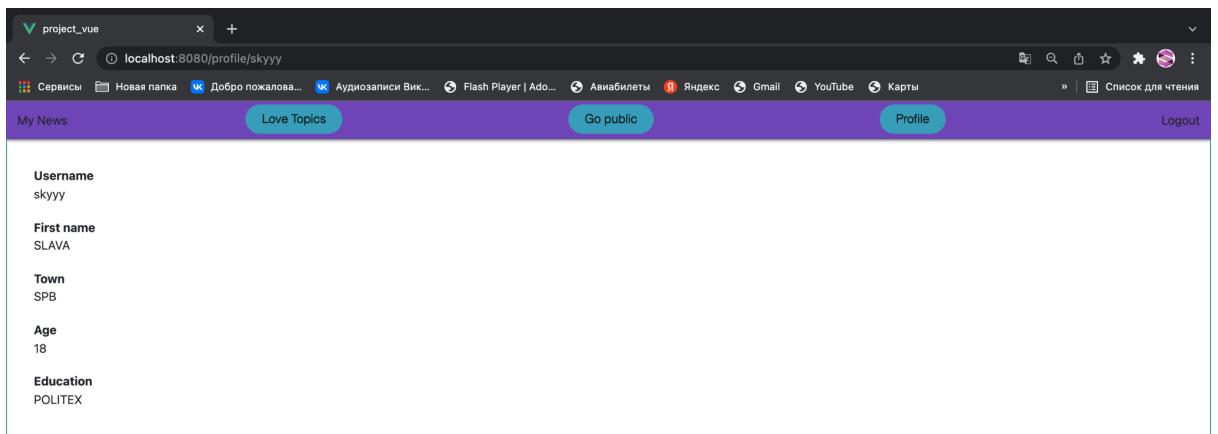
- 1
Owner:skylar
Title:fsd
Body:dsf
Tags:sdf
- 2
Owner:skylar
Title:fsdfs
Body:dsfsf
Tags:sdfs
- 3
Owner:skyyy
Title:Hello
Body:world
Tags:OK
- 4
Owner:skylar_rr
Title:ITMO
Body:ONE LOVE
Tags:DL тож люблю



6. Профайл



7. Профайл другого пользователя



ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было реализовано веб-приложения для записи и размещения личных новостей. Получил практические знания и опыт в работе с backend частью, а именно реализацию серверной части средствами DjangoREST framework, контейнеризации Docker и развертки в нем серверной части приложения. Также получил дополнительные знания в ООП и языке Python и применил уже имеющиеся на практике. Научился выстраивать архитектуру приложения и реализовывать интерфес (fronted часть приложения) средствами Vue.js.

СПИСОК ЛИТЕРАТУРЫ

1. Документация Django [Электронный ресурс] —
<https://docs.djangoproject.com/en/3.0/>. Дата обращения 10.01.2022.
2. Документация Django REST Framework [Электронный ресурс] —
<https://www.django-rest-framework.org>. Дата обращения 10.01.2022.
3. Фильтрация средствами Django REST Framework [Электронный ресурс] —
<https://www.django-rest-framework.org/api-guide/filtering/>. Дата обращения 10.01.2022.
4. Документация Vue.js [Электронный ресурс] — <https://vuejs.org/v2/guide/> Дата обращения 10.01.2022.