Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Программная инженерия
кафедра

КУРСОВОЙ ПРОЕКТ

Разработка веб-приложения для учета тренировок на основе Django
тема проекта

Руководитель _____ П.В.Пересунько
подпись, дата — инициалы, фамилия

Студент _____ _____ _____
номер группы, зачётной книжки — подпись, дата — инициалы, фамилия

Студент КИ23-17/1Б, 032318988 _____ Александров Е.А.
номер группы, зачётной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

Подп. и дата
Взам. инв. №
Инв. № дубл.
Подп. и дата
Инв. № подл.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В условиях стремительного развития цифровых технологий и повсеместного интереса к здоровому образу жизни, эффективные инструменты для систематизации физических нагрузок становятся критически важными. Особенно актуальна эта потребность для широкой аудитории — от новичков, до опытных спортсменов. Существующие решения часто страдают от фрагментированности данных, недостаточной персонализации или ограничений мобильными платформами. Целью данного проекта стало создание веб-приложения "Трекер тренировок", призванного решить эти проблемы через универсальную и доступную платформу для планирования, фиксации и анализа тренировочного процесса.

Актуальность работы обусловлена тем, что трекер тренировок дает возможность фиксировать результаты и наглядно отслеживать прогресс, кроме того обеспечивая кроссплатформенность и мгновенную доступность без установки приложений, что критически важно для поддержания вовлечённости пользователей и предотвращения «выгорания».

В ходе работы была поставлена задача спроектировать и реализовать полнофункциональное клиент-серверное приложение. Для серверной разработки использовался Python[1] с фреймворком Django[2], и Django REST Framework[3]. Клиентская часть реализована на JavaScript[4] с библиотекой React.js[5]. Методы реализации включали модульную разработку по принципам Agile[6], разбитую на семь спринтов.

Новизна проекта заключается в использовании связки Django-REST-React для задач фитнес-трекинга, таких как эффективное хранение тренировочных данных и быстрый доступ к ним. Работа демонстрирует практическое применение современных веб-технологий для решения актуальной задачи в сфере здоровья. Результатом является работоспособное веб-приложение, готовое к использованию и демонстрации на защите, с потенциалом для дальнейшего развития.

1 Проектирование

Сценарии использования системы:

- добавление тренировки. Пользователь нажимает «Добавить тренировку», указывает длительность (например, 60 минут). Система сохраняет данные;
- добавление упражнения. В созданной тренировке пользователь выбирает «Добавить упражнение», вводит название, количество подходов, повторений, рабочий вес и время отдыха (например, "Жим лежа: 4x10, 80 кг, отдых 90 сек"). Система сохраняет упражнение;
- просмотр тренировок. Пользователь открывает главную страницу где видит список всех занятий с датами и длительностью;
- просмотр упражнений в тренировке. Выбрав конкретную тренировку, пользователь видит список всех добавленных упражнений с указанными параметрами (подходы, вес, отдых);
- регистрация нового пользователя. Неавторизованный пользователь заполняет форму (логин, пароль), после чего получает доступ к системе;
- вход в аккаунт. Зарегистрированный пользователь вводит свои данные для авторизации и доступа к своим тренировкам.

Диаграмма вариантов использования представлена на рисунке 1.

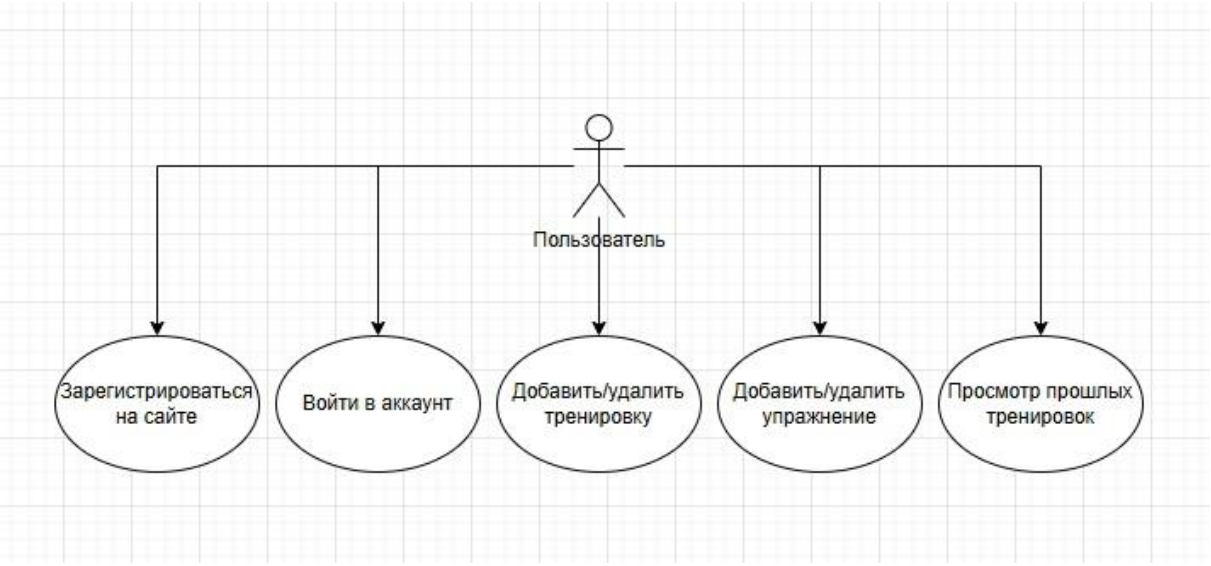

Рисунок 1 – Диаграмма вариантов использования

Диаграмма деятельности представлена на рисунке 2.

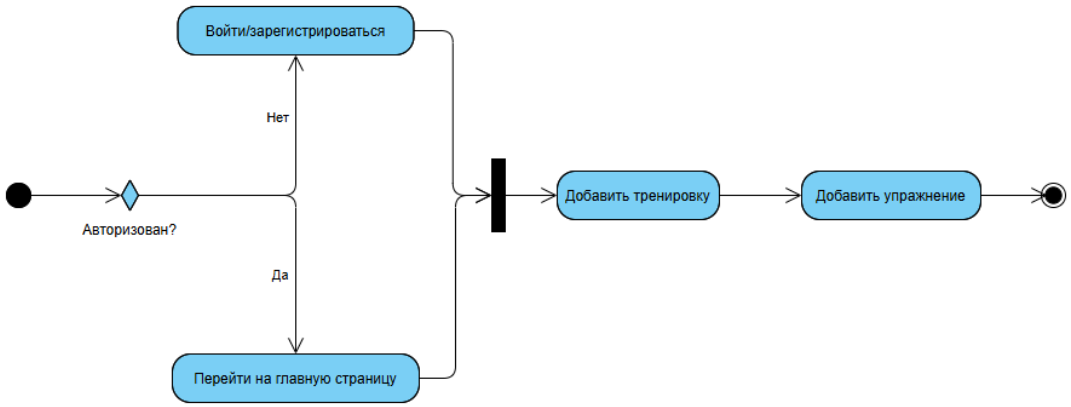

Рисунок 2 – Диаграмма вариантов использования

Шаблоны проектирования, использующиеся в приложении:

- MVC(Model-View-Controller). Используется по умолчанию в Django;
- Singleton. Используется в Django ORM для подключения к базе данных;
- Observer. Применяется при использовании Django signals для изменения статуса пользователя после подтверждения аккаунта;
- Decorator. Используется для проверки аутентификации.

2 Разработка минимального стоящего продукта

Минимально стоящий продукт (MVP) трекера тренировок представляет из себя веб-приложение с ограниченным функционалом.

Он включает в себя основные функции:

- добавление/удаление тренировок
- регистрация/вход в аккаунт
- добавление/удаление упражнений
- просмотр тренировок

2.1 Первый спринт

Задачи на первый спринт представлены в таблице 1.

Таблица 1 – Задачи первого спринта

№	Оценка	Задача
1	2	Создание сервиса для регистрации и просмотра пользователей
2	3	Создание и настройка Docker-compose[7] файла для взаимодействия сервиса с базой данных Postgresql[8]
3	3	Добавление функциональности регистрации и входа в приложение
4	5	Добавление функциональности верификации Email при регистрации.
5	3	Добавление функциональности просмотра списка пользователей, удаления их и просмотра деталей по каждому пользователю
sum	16	

2.1.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 3 по 8.

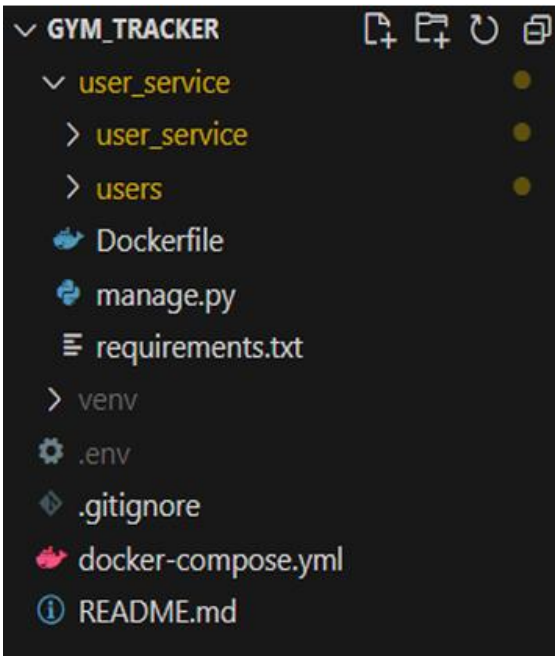

Рисунок 3 - Структура проекта

```yaml
version: '3.9'

services:
  db:
    image: postgres:15.1-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
  user-service:
    image: user-service
    build:
      context: ./user_service/
      dockerfile: Dockerfile
    init: true
    volumes:
      - ./user_service/:/usr/src/app/
    ports:
      - "8000:8000"
    depends_on:
      - db
    env_file:
      - .env
volumes:
  postgres_data:
```

Рисунок 4 – Docker-compose файл

Рисунок 5 - API Endpoint для регистрации и входа в приложение

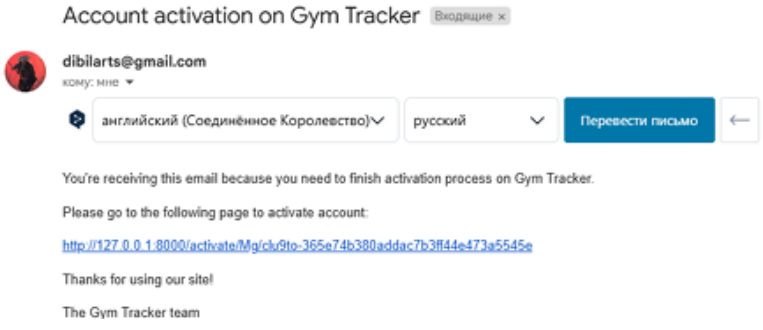

Рисунок 6 - Пример верификационного письма

Рисунок 7 – Список пользователей

Рисунок 8 – Диаграмма сгорания задач

2.2 Второй спринт

Задачи на второй спринт представлены в таблице 2.

Таблица 2 – Задачи второго спринта

№	Оценка	Задача
1	4	Реализация функционала сброса и восстановления пароля
2	5	Реализация отправки Email, используя Celery[9] и RabbitMQ[10]
3	3	Написание тестов с помощью Unittest[11] для всего функционала
sum	12	

2.2.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 9 по 16.

```
+    @action(["post"], detail=False)
+    def reset_password(self, request, *args, **kwargs):
+        serializer = self.get_serializer(data=request.data)
+        serializer.is_valid(raise_exception=True)
+
+        user = serializer.get_user()
+        if user:
+            reset_code = self.generate_reset_code()
+
+            PasswordReset.objects.create(
+                email=user.email,
+                reset_code=reset_code,
+                expires_at=timezone.now() + timedelta(minutes=10)
+            )
+
+            message = json.dumps({
+                'email': user.email,
+                'reset_code': reset_code,
+            })
+
+            self.send_to_rabbitmq(message)
+
+        return Response(status=status.HTTP_204_NO_CONTENT)
+
+    def generate_reset_code(self):
+        code = ''.join(choices("0123456789", k=6))
+        return code
+
+    @action(["post"], detail=False)
+    def reset_password_confirm(self, request, *args, **kwargs):
+        email = request.data.get('email')
+        new_password = request.data.get('new_password')
+        reset_code = request.data.get('reset_code')
+
+        try:
```

Рисунок 9 – Реализация представления для сброса пароля

```
+
+        try:
+            reset_entry = PasswordReset.objects.get(email=email, reset_code=reset_code)
+            if reset_entry.is_expired():
+                return Response({'error': 'Code expired'}, status=status.HTTP_400_BAD_REQUEST)
+
+            user = User.objects.filter(email=email).first()
+            if user:
+                user.set_password(new_password)
+                user.save()
+
+                return Response(status=status.HTTP_204_NO_CONTENT)
+
+            return Response({'error': 'User is not found'}, status=status.HTTP_404_NOT_FOUND)
+
+        except PasswordReset.DoesNotExist:
+            return Response({'error': 'Code is not right'}, status=status.HTTP_400_BAD_REQUEST)
```

Рисунок 10 – Реализация представления для сброса пароля

notification_service/notifications/tasks.py

```
@@ -0,0 +1,21 @@
+ from celery import shared_task
+ from django.core.mail import send_mail
+
+ @shared_task
+ def send_activation_email(email, activation_link):
+     print('Sending')
+     send_mail(
+         subject='Activation on Gym Tracker!',
+         message=f'Hello, to activate your account, go through the following link: {activation_link}',
+         from_email='dibilarts@gmail.com',
+         recipient_list=[email]
+     )
+
+ @shared_task
+ def send_password_reset_email(email, reset_code):
+     send_mail(
+         subject='Password reset on Gym Tracker!',
+         message=f'Your reset code: {reset_code}',
+         from_email='dibilarts@gmail.com',
+         recipient_list=[email],
+     )
```

Рисунок 11 – Реализация отправки Email через Celery

user_service/users/tests/test_models.py

```
@@ -0,0 +1,30 @@
+ from django.test import TestCase
+ from ..models import CustomUser, PasswordReset
+ from rest_framework.authtoken.models import Token
+ from django.utils import timezone
+
+ class TestCustomUser(TestCase):
+     def test_user_creation(self):
+         user = CustomUser.objects.create(username='test', password='Jojojo123')
+
+         self.assertIsInstance(user, CustomUser)
+         self.assertFalse(user.is_active)
+
+ class TestAuthTokenCreation(TestCase):
+     def test_auth_token_creation(self):
+         user = CustomUser.objects.create(username='test', password='Jojojo123')
+
+         self.assertTrue(Token.objects.filter(user=user).exists())
+
+ class TestPasswordReset(TestCase):
+     def setUp(self):
+         self.password_reset = PasswordReset.objects.create(
+             email = 'test@gmail.com',
+             reset_code = '111111',
+             expires_at = timezone.now() + timezone.timedelta(minutes=10)
+         )
+
+     def test_expire_time(self):
+         self.assertFalse(self.password_reset.is_expired())
+         self.password_reset.expires_at = self.password_reset.created_at
+         self.assertTrue(self.password_reset.is_expired())
```

Рисунок 12 – Реализация тестов

```
@@ -0,0 +1,154 @@
+ from django.test import TestCase
+ from django.urls import reverse
+ from rest_framework import status
+ from django.utils.http import urlsafe_base64_encode
+ from django.utils.encoding import force_bytes
+ from django.contrib.auth.tokens import default_token_generator
+ from rest_framework.authtoken.models import Token
+ from ..models import CustomUser, PasswordReset
+ from rest_framework_simplejwt.tokens import RefreshToken
+ from unittest.mock import patch
+ from django.utils import timezone
+ import json
+
+
+ class TestUserViewSet(TestCase):
+     def setUp(self):
+         self.url = reverse('customuser-list')
+
+     def test_creation_forbidden(self):
+         response = self.client.post(self.url, {})
+         self.assertEqual(response.status_code, status.HTTP_403_FORBIDDEN)
+         self.assertEqual(response.data, {"detail": "Creation is not allowed."})
+
+
+ class TestUserActivationView(TestCase):
+     def setUp(self):
+         self.user = CustomUser.objects.create_user(
+             username='testuser',
+             password='password'
+         )
+         self.uid = urlsafe_base64_encode(force_bytes(self.user.id))
+         self.token = default_token_generator.make_token(self.user)
+         self.url = reverse('user-activate', args=[self.uid, self.token])
```

Рисунок 13 – Реализация тестов

```
+    def test_activation_success(self):
+        response = self.client.get(self.url)
+        self.assertEqual(response.status_code, status.HTTP_200_OK)
+        self.user.refresh_from_db()
+        self.assertTrue(self.user.is_active)
+
+    def test_activation_user_not_found(self):
+        invalid_url = reverse('user-activate', args=['nothing', 'invalid'])
+        response = self.client.get(invalid_url)
+        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
+
+    def test_activation_url_invalid(self):
+        invalid_url = reverse('user-activate', args=[self.uid, 'invalid'])
+        response = self.client.get(invalid_url)
+        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
+
+class TestCustomUserViewSet(TestCase):
+    def setUp(self):
+        self.registration_url = reverse('auth-list')
+        self.password_reset_url = reverse('auth-reset-password')
+        self.user = CustomUser.objects.create_user(
+            email='test@example.com',
+            username='testuser',
+            password='password'
+        )
+        self.user.is_active = True
+        self.user.save()
+        self.reset_code = '123456'
+        self.password_reset = PasswordReset.objects.create(
+            email=self.user.email,
+            reset_code=self.reset_code,
+            expires_at=timezone.now() + timezone.timedelta(minutes=10)
+        )
```

Рисунок 14 – Реализация тестов

```
+    @patch('users.views.CustomUserViewSet.send_to_rabbitmq')
+    def test_sending_activation_message(self, mock_send):
+        user_data = {
+            "username": "new_user",
+            "password": "new_password",
+            "email": "new@example.com"
+        }
+        response = self.client.post(self.registration_url, user_data, format='json')
+        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
+        self.assertTrue(mock_send.called)
+
+        args = mock_send.call_args[0]
+        message = json.loads(args[0])
+        self.assertEqual(message['email'], "new@example.com")
+        self.assertIn('activation_link', message)
+
+    @patch('users.views.CustomUserViewSet.send_to_rabbitmq')
+    def test_password_reset_code_sending(self, mock_send):
+        user_data = {
+            "email": "test@example.com"
+        }
+        response = self.client.post(self.password_reset_url, user_data, format='json')
+        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
+        self.assertTrue(mock_send.called)
+
+        args = mock_send.call_args[0]
+        message = json.loads(args[0])
+        self.assertEqual(message['email'], "test@example.com")
+        self.assertIn('reset_code', message)
+
+    @patch('users.views.CustomUserViewSet.send_to_rabbitmq')
+    def test_password_reset_code_sending_user_not_found(self, mock_send):
+        user_data = {
+            "email": "nan@example.com"
+        }
```

Рисунок 15 – Реализация тестов

Рисунок 16 – Диаграмма сгорания задач

2.3 Третий спринт

Задачи на третий спринт представлены в таблице 3.

Таблица 3 – Задачи третьего спринта

№	Оценка	Задача
1	3	Реализация CRUD training service
2	4	Реализация взаимодействия с user service
3	3	Написание тестов для всего функционала
4	2	Реализация авторизации через Google
sum	12	

2.3.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 17 по 23.

user_service/user_service/settings.py

```
+ AUTH_USER_MODEL = 'users.CustomUser'
+
+ CORS_ORIGIN_ALLOW_ALL = True
+
+ SITE_ID = 2
+
+ SOCIALACCOUNT_PROVIDERS = {
+     'google': {
+         'SCOPE': [
+             'profile',
+             'email',
+         ],
+        'AUTH_PARAMS': {'access_type': 'online'}
+     }
+ }
+
+ AUTHENTICATION_BACKENDS = (
+     'django.contrib.auth.backends.ModelBackend',
+     'allauth.account.auth_backends.AuthenticationBackend',
+ )
+
+ LOGIN_REDIRECT_URL = "/"
+ LOGOUT_REDIRECT_URL = "/"
+
```

user_service/user_service/urls.py

```
@@ -9,4 +9,5 @@
     path("api/auth/", include('djoser.urls.authtoken')),
     path("api/", include(users_urls)),
     path('api/auth/activate/<uid>/<token>/', users_views.UserActivationView.as_view(), name='user-activate'),
+    path('api/auth/o/', include('allauth.urls')),
 ]
```

Рисунок 17 – Реализация авторизации через Google

training_service/trainings/tests/test_models.py

```
@@ -0,0 +1,23 @@
+ from django.test import TestCase
+ from ..models import Training, Exercise
+
+ class TestTraining(TestCase):
+     def test_training_creation(self):
+         training = Training.objects.create(
+             user_id=1
+         )
+         self.assertTrue(isinstance(training, Training))
+
+ class TestExercise(TestCase):
+     def setUp(self):
+         self.training = Training.objects.create(user_id=1)
+
+     def test_exercise_creation(self):
+         exercise = Exercise.objects.create(
+             name="Test Exercise",
+             weight=50,
+             reps=10,
+             sets=5,
+             training_id=self.training
+         )
+         self.assertTrue(isinstance(exercise, Exercise))
```

Рисунок 18 – Реализация тестов

training_service/trainings/tests/test_views.py

```
@@ -0,0 +1,65 @@
+ from django.test import TestCase
+ from django.urls import reverse
+ from rest_framework import status
+ from ..models import Training, Exercise
+
+
+ class TestTrainingViewSet(TestCase):
+     def setUp(self):
+         self.url = reverse('training-list')
+         self.training = Training.objects.create(
+             user_id=1,
+             duration=10
+         )
+
+     def test_get_trainings(self):
+         response = self.client.get(self.url, {})
+         self.assertEqual(response.status_code, status.HTTP_200_OK)
+
+     def test_get_trainings_by_user_id(self):
+         response = self.client.get(self.url, {'user_id': 1})
+         self.assertEqual(response.status_code, status.HTTP_200_OK)
+         self.assertEqual(len(response.data), 1)
+
+     def test_create_training(self):
+         data = {
+             "user_id": 1,
+             "duration": 15
+         }
+         response = self.client.post(self.url, data, format='json')
+         self.assertEqual(response.status_code, status.HTTP_201_CREATED)
+         self.assertEqual(Training.objects.count(), 2)
+
+     def test_create_training_user_not_found(self):
+         data = {
+             "user_id": 999,
+             "duration": 15
+         }
+         response = self.client.post(self.url, data, format='json')
```

Рисунок 19 – Реализация тестов

```
+
+ class TestExerciseViewSet(TestCase):
+     def setUp(self):
+         self.url = reverse('exercise-list')
+         self.training = Training.objects.create(
+             user_id=1,
+             duration=10
+         )
+         self.exercise = Exercise.objects.create(
+             name="Test Exercise",
+             weight=50,
+             reps=10,
+             sets=5,
+             training_id=self.training
+         )
+
+     def test_get_exercise(self):
+         response = self.client.get(self.url, {})
+         self.assertEqual(response.status_code, status.HTTP_200_OK)
+
+     def test_get_exercise_by_training_id(self):
+         response = self.client.get(self.url, {'training_id': self.training.id})
+         self.assertEqual(response.status_code, status.HTTP_200_OK)
+         self.assertEqual(len(response.data), 1)
+
```

Рисунок 20 – Реализация тестов

Training List

OPTIONS GET

GET /api/trainings/

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "user_id": 1,
        "date": "2025-03-23",
        "duration": 15
    },
    {
        "id": 2,
        "user_id": 1,
        "date": "2025-03-23",
        "duration": 15
    },
    {
        "id": 3,
        "user_id": 45,
        "date": "2025-03-24",
        "duration": 45
    }
]
```

Рисунок 21 – Реализация Training Service

```python
class TrainingViewSet(ModelViewSet):
    permission_classes = [AllowAny]
    queryset = Training.objects.all()
    serializer_class = TrainingSerializer

    def create(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        try:
            self.perform_create(serializer)
            headers = self.get_success_headers(serializer.data)
            return Response(serializer.data, status=status.HTTP_201_CREATED, headers=headers)
        except Exception as e:
            return Response(status=status.HTTP_400_BAD_REQUEST, data={"Error": str(e)})

    def get_queryset(self, *args, **kwargs):
        if self.request.query_params.get('user_id'):
            return Training.objects.filter(user_id=self.request.query_params.get('user_id'))
        return Training.objects.all()

    def perform_create(self, serializer):
        user_id = self.request.data.get('user_id')

        response = requests.get(f'http://user-service:8000/api/users/{user_id}/')

        if response.status_code == 200:
            if response.json().get('is_active'):
                serializer.save()
            else:
                raise Exception('User is not active')
        else:
            raise Exception('User not found')
```

Рисунок 22 – Реализация связи с user service

Рисунок 23 – Диаграмма сгорания задач

2.4 Четвертый спринт

Задачи на четвертый спринт представлены в таблице 4.

Таблица 4 – Задачи четвертого спринта

№	Оценка	Задача
1	5	Реализация документации api с помощью Swagger[12]
2	5	Реализация api gateway через Nginx[13]
3	3	Написание интеграционных тестов
sum	13	

2.4.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 24 по 27.

```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;
    keepalive_timeout 65;

    server {
        listen 0.0.0.0:80;

        location /api/users/ {
            proxy_pass http://user-service:8000/api/users/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /api/trainings/ {
            proxy_pass http://training-service:8000/api/trainings/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /api/auth/ {
            proxy_pass http://user-service:8000/api/auth/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /api/exercises/ {
            proxy_pass http://training-service:8000/api/exercises/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Рисунок 24 – Конфигурация Nginx

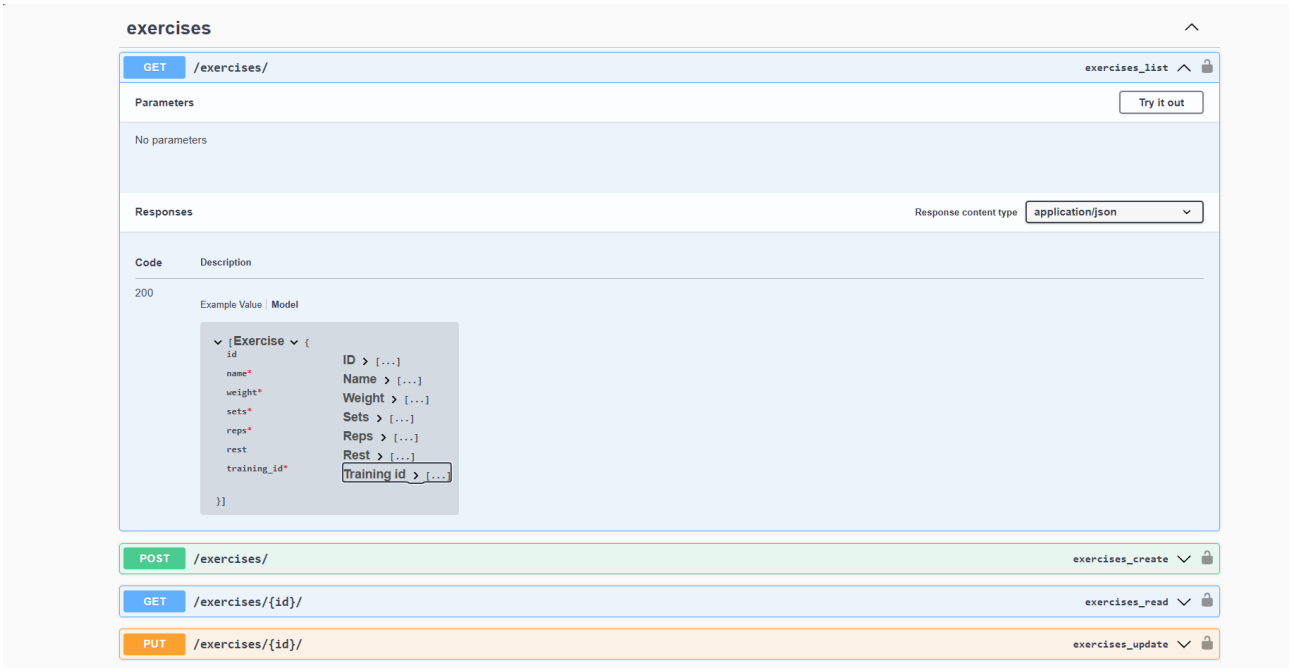

Рисунок 25 – Документация api через Swagger

```python
@patch('users.views.CustomUserViewSet.send_to_rabbitmq')
def test_sending_activation_message(self, mock_send):
    user_data = {
        "username": "new_user",
        "password": "new_password",
        "email": "new@example.com"
    }
    response = self.client.post(self.registration_url, user_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertTrue(mock_send.called)

    args = mock_send.call_args[0]
    message = json.loads(args[0])
    self.assertEqual(message['email'], "new@example.com")
    self.assertIn('activation_link', message)

@patch('users.views.CustomUserViewSet.send_to_rabbitmq')
def test_password_reset_code_sending(self, mock_send):
    user_data = {
        "email": "test@example.com"
    }
    response = self.client.post(self.password_reset_url, user_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
    self.assertTrue(mock_send.called)

    args = mock_send.call_args[0]
    message = json.loads(args[0])
    self.assertEqual(message['email'], "test@example.com")
    self.assertIn('reset_code', message)

@patch('users.views.CustomUserViewSet.send_to_rabbitmq')
def test_password_reset_code_sending_user_not_found(self, mock_send):
    user_data = {
        "email": "nan@example.com"
    }
    response = self.client.post(self.password_reset_url, user_data, format='json')
    self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)
    self.assertFalse(mock_send.called)
```

Рисунок 26 – Тесты

Рисунок 27 – Диаграмма сгорания задач

3 Подготовка релиза

За предыдущие четыре спринта были реализованы ключевые функции, позволяющие проходить основные сценарии. Далее следует добавление новых функций и загрузка на сервер.

3.1 Пятый спринт

Задачи на пятый спринт представлены в таблице 5.

Таблица 5 – Задачи пятого спринта

№	Оценка	Задача
1	4	Реализация интерфейса веб-приложения
2	5	Реализация связи клиентской части веб-приложения с серверной частью
3	3	Настройка Nginx для сервера
sum	12	

3.1.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 28 по 30.

```
location / {
    root /usr/share/nginx/html/frontend;
    try_files $uri $uri/ /index.html;
}

 location /static/ {
    alias /usr/share/nginx/html/frontend/static/;
    expires 30d;
    add_header Cache-Control "public";
}

location /backend-static/ {
    alias /usr/share/nginx/html/static/;
    expires 30d;
    add_header Cache-Control "public";
}
```

Рисунок 28 – Конфигурация Nginx

Рисунок 29 – Интерфейс для связи с серверной частью

Рисунок 30 – Диаграмма сгорания задач

3.2 Шестой спринт

Задачи на шестой спринт представлены в таблице 6.

Таблица 6 – Задачи шестого спринта

№	Оценка	Задача
1	5	Реализация дизайна веб-приложения
2	5	Настройка системы логирования Prometheus[14]
3	4	Настройка системы мониторинга Grafana[15]
sum	14	

3.2.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 31 по 34.

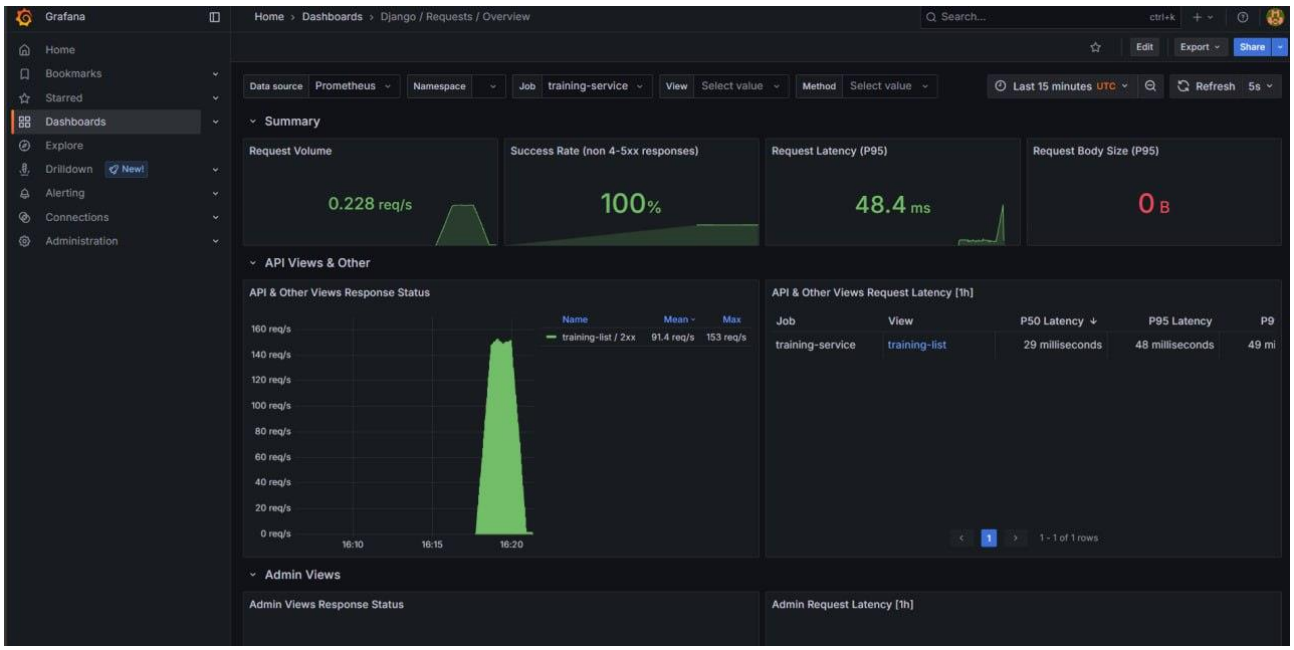

Рисунок 31 – Система мониторинга Grafana

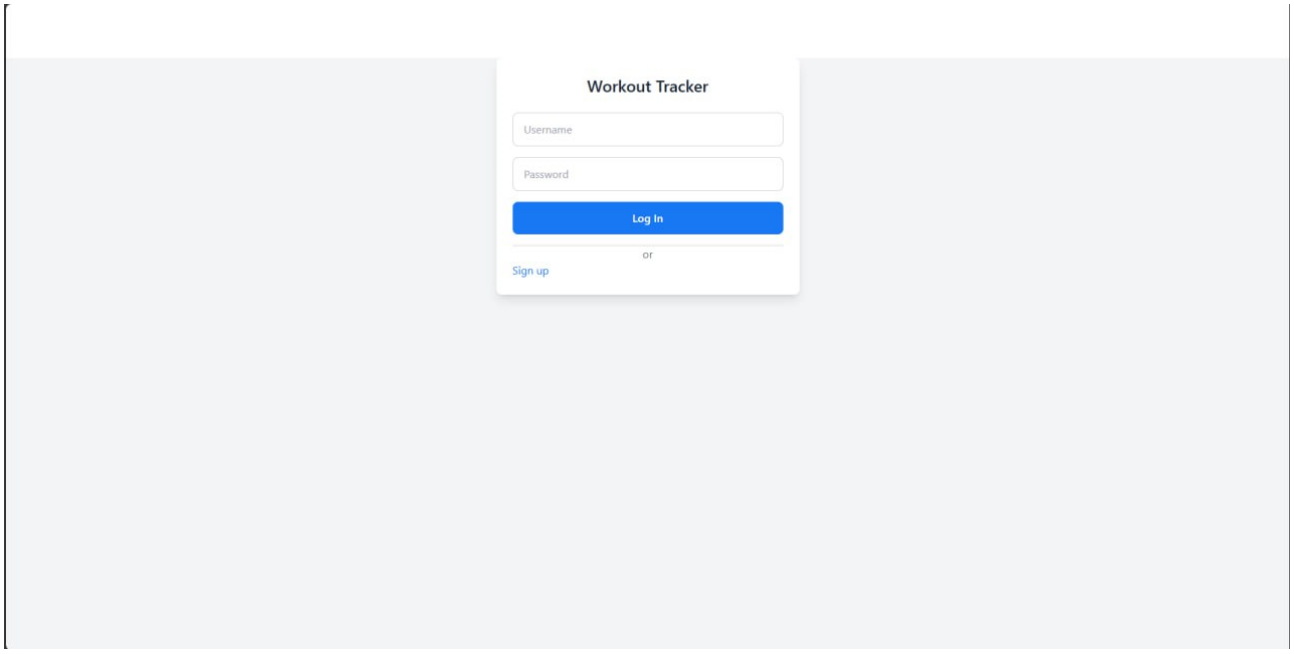

Рисунок 32 – Дизайн интерфейса

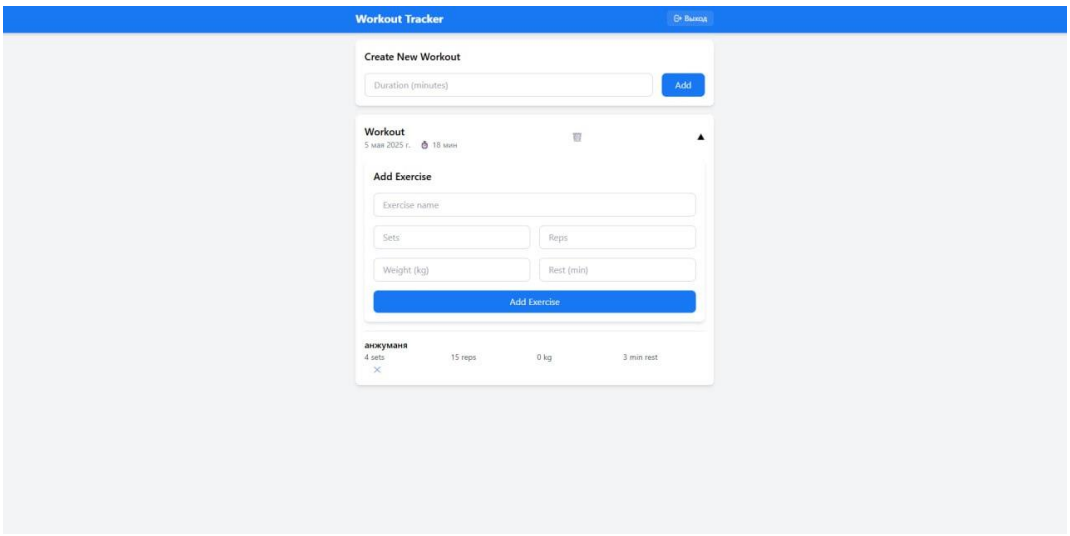

Рисунок 33 – Дизайн интерфейса

Рисунок 34 – Диаграмма сгорания задач

3.3 Седьмой спринт

Задачи на седьмой спринт представлены в таблице 7.

Таблица 7 – Задачи седьмого спринта

№	Оценка	Задача
1	3	Исправление багов
2	5	Рефакторинг кода
3	5	Деплой веб-приложения
sum	13	

3.3.1 Результаты выполнения задач

Результаты выполнения задач представлены на рисунках с 35 по 37.

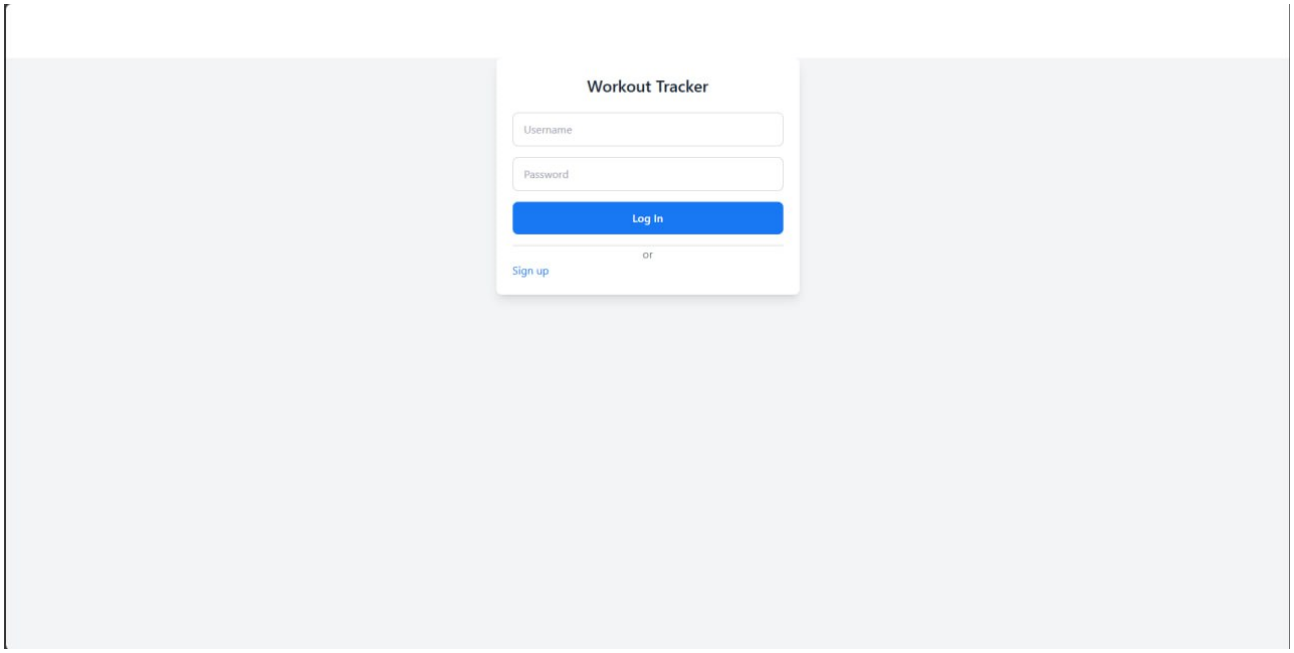

Рисунок 35 – Результат деплоя

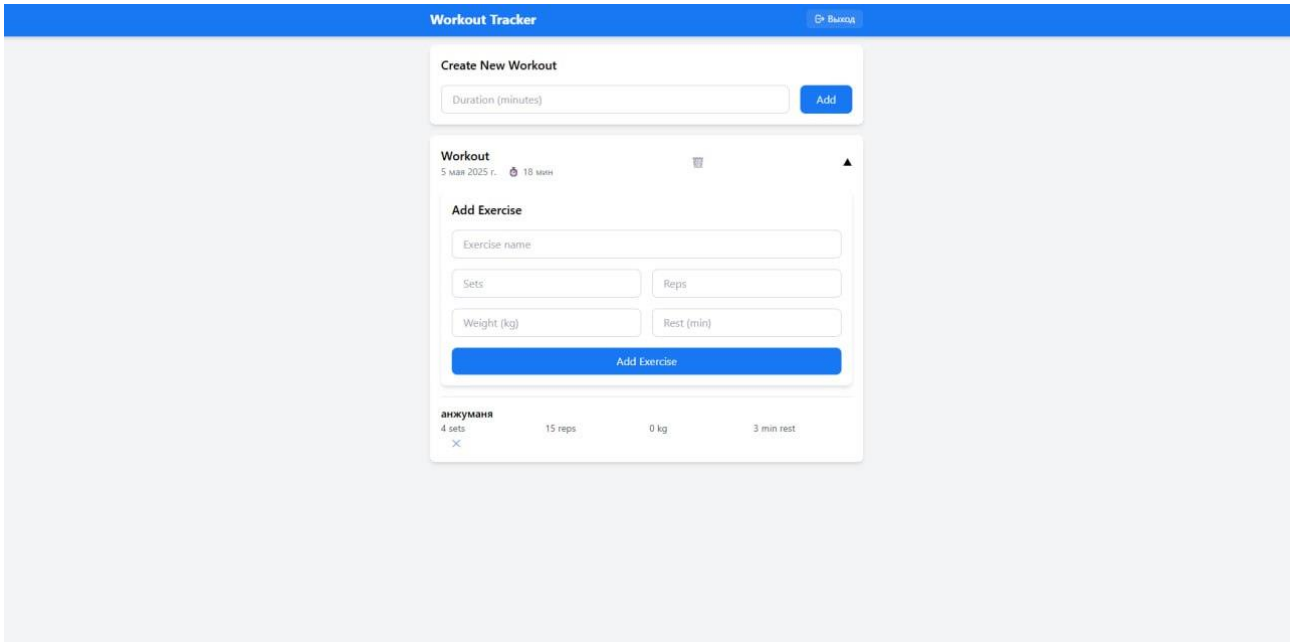

Рисунок 36 – Результат деплоя

Рисунок 37 – Диаграмма сгорания задач

4 Инструкция по использованию

Для того чтобы воспользоваться сайтом необходимо открыть любой веб-браузер и перейти на URL-адрес http://87.228.83.10/, затем пройти регистрацию и войти в аккаунт. Далее будет доступна возможность добавления и редактирования тренировок.

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта поэтапно реализован полноценный веб-сервис для учета тренировок, демонстрирующий комплексное применение современных технологий и архитектурных подходов. Каждый спринт последовательно решал ключевые задачи: от развертывания инфраструктуры с использованием Docker и настройки взаимодействия сервисов через API Gateway на Nginx и очереди RabbitMQ до реализации бизнес-логики и обеспечения отказоустойчивости.

Следующим критически важным шагом стала разработка тренировочного микросервиса с полным циклом операций (создание, просмотр, обновление и удаление тренировок и упражнений), включая взаимодействие с пользовательским сервисом. Для обеспечения прозрачности и надежности системы внедрена многоуровневая система тестирования: юнит-тесты для модулей, интеграционные тесты для проверки взаимодействия сервисов, а также сквозное тестирование API через Swagger-документацию. Особое внимание уделено промышленным требованиям: реализовано централизованное логирование, мониторинг метрик производительности через Grafana, а также рефакторинг кода для соответствия стандартам качества.

Финальные этапы включали разработку клиентского интерфейса на React.js с адаптивным дизайном, обеспечивающим удобство работы на любых устройствах, и его интеграцию с бэкендом на Django через REST API. После оптимизации конфигурации Nginx и устранения выявленных багов проект был успешно развернут в продуктивной среде. Тестирование подтвердило стабильность работы всех компонентов: от корректности отправки email до отказоустойчивости при высоких нагрузках. Таким образом, поставленная цель достигнута — создано масштабируемое веб-приложение, готовое к промышленной эксплуатации. Проект не только отражает освоение технологического стека (Django, DRF, Celery, Docker), но и демонстрирует навыки построения распределенных систем, что закладывает фундамент для дальнейшего расширения функционала.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Python [Электронный ресурс]. - Режим доступа: https://www.python.org/doc/ (Дата обращения: 27.06.2024)

2. Документация Django [Электронный ресурс]. - Режим доступа: https://docs.djangoproject.com/en/5.2/ (Дата обращения: 27.06.2024)

3. Документация Django REST Framework [Электронный ресурс]. - Режим доступа: https://www.django-rest-framework.org/tutorial/quickstart/ (Дата обращения: 27.06.2024)

4. Документация JavaScript [Электронный ресурс]. - Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript (Дата обращения: 27.06.2024)

5. Документация React.js [Электронный ресурс]. - Режим доступа: https://react.dev/learn (Дата обращения: 27.06.2024)

6. Статья "Agile Software Development"[Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Agile_software_development (Дата обращения: 27.06.2024)

7. Документация Docker [Электронный ресурс]. - Режим доступа: https://docs.docker.com/ (Дата обращения: 27.06.2024)

8. Документация PostgreSQL [Электронный ресурс]. - Режим доступа: https://www.postgresql.org/docs/ (Дата обращения: 27.06.2024)

9. Документация Celery [Электронный ресурс]. - Режим доступа: https://docs.celeryq.dev/en/stable/index.html (Дата обращения: 27.06.2024)

10. Документация RabbitMQ [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/docs (Дата обращения: 27.06.2024)

11. Документация модуля unittest [Электронный ресурс]. - Режим доступа: https://docs.python.org/3/library/unittest.html (Дата обращения: 27.06.2024)

12. Спецификация Swagger [Электронный ресурс]. - Режим доступа: https://docs.swagger.io/spec.html (Дата обращения: 27.06.2024)

13. Документация Nginx [Электронный ресурс]. - Режим доступа: https://nginx.org/en/docs/ (Дата обращения: 27.06.2024)

14. Документация Prometheus [Электронный ресурс]. - Режим доступа: https://prometheus-docs.netlify.app/docs/prometheus/latest/getting_started/ (Дата обращения: 27.06.2024)

15. Документация Grafana [Электронный ресурс]. - Режим доступа: https://grafana.com/docs/ (Дата обращения: 27.06.2024)

ПРИЛОЖЕНИЕ А
(рекомендуемое)

Отрисовка UserFlow

User Flow для сайта (рисунок А.1).

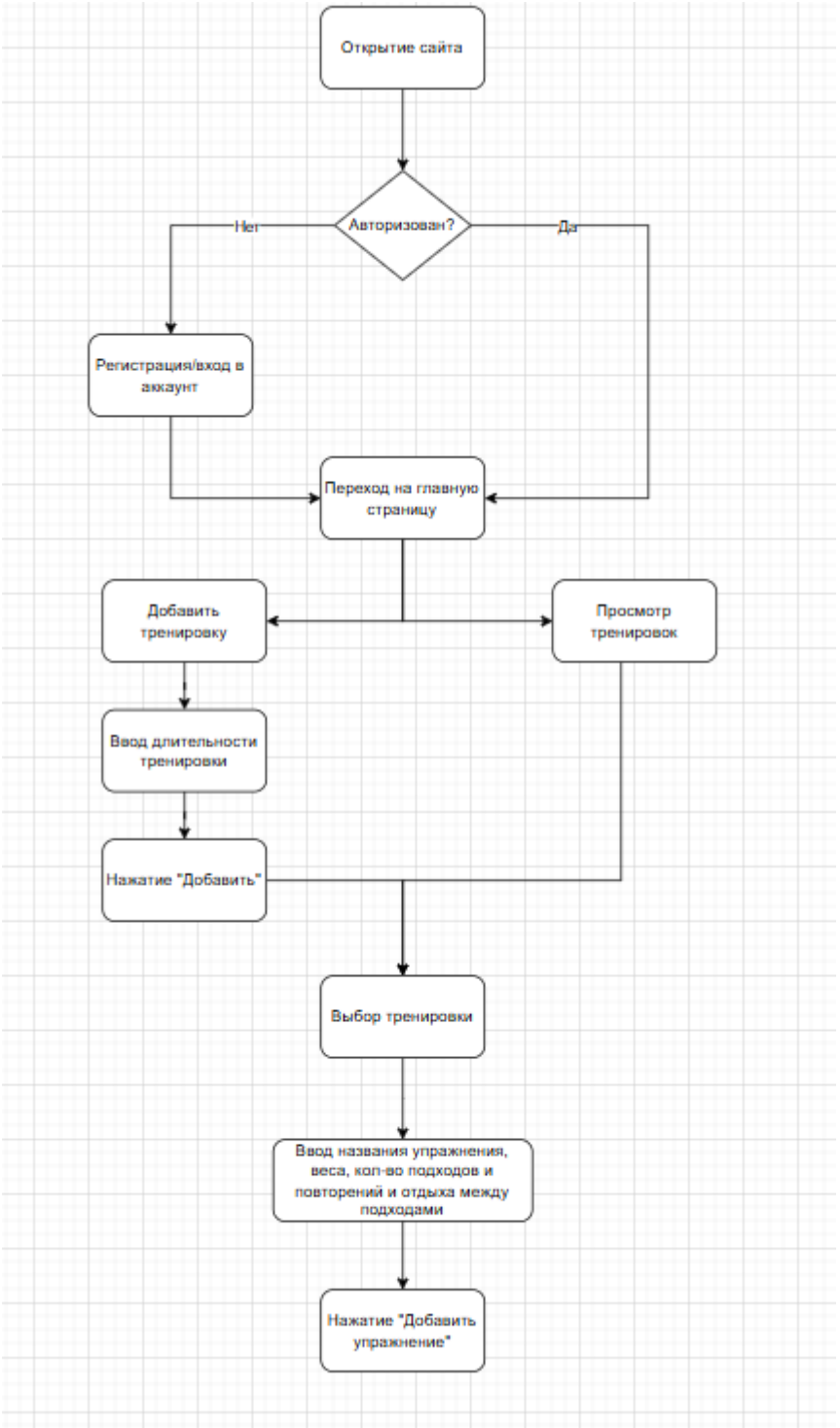

Рисунок А.1 – User Flow

ПРИЛОЖЕНИЕ Б
(рекомендуемое)

Диаграмма окон

Диаграмма окон для сайта (рисунок Б.1).

Рисунок Б.1 – Диаграмма окон

ПРИЛОЖЕНИЕ В
(рекомендуемое)

Диаграмма последовательности

Диаграмма последовательности для сайта (рисунок В.1).

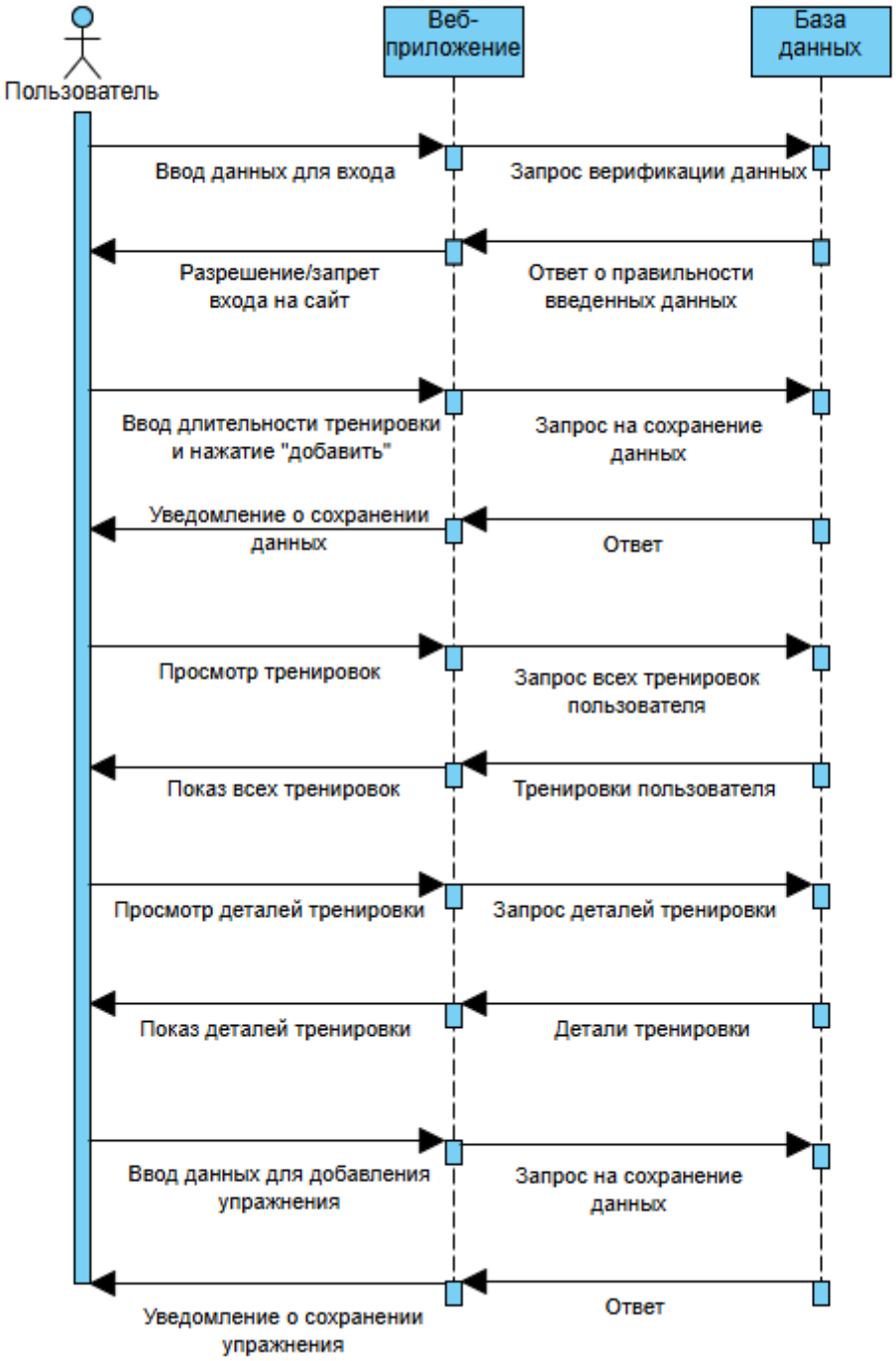

Рисунок В.1 – Диаграмма последовательности

Лист регистрации изменений

Изм	Новых				Всего листов (страниц)	№ документа	Входящий № сопроводительного документа и	Подпись	Дата
	Измененных	Замененных	Новых	Изъятых					

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

Инв. № подл

Ли	Изм.	№ докум.	Подп.	Дата	КП-090304-2024 ПЗ	Лист
						30