Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет программной инженерии и компьютерной техники

«Информационные системы и базы данных»

Курсовая работа, этап №4

Выполнили студенты группы P33111

Окладников Константин

Файзулин Адиль

Преподаватель:

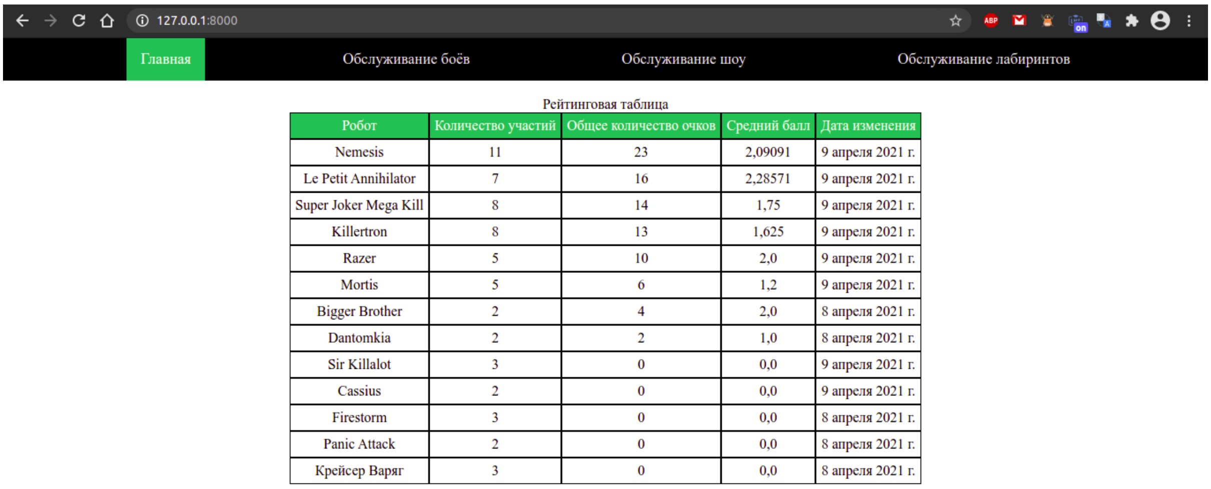
Харитонова Анастасия Евгеньевна

Санкт-Петербург

2021

Снимки главного бизнес процесса в реализации веб-приложения

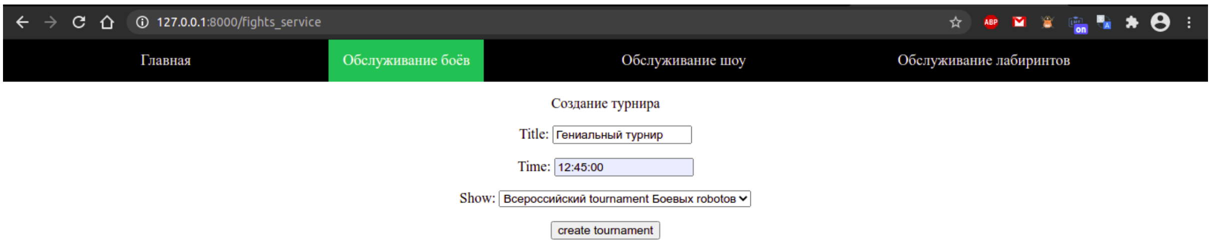
Главная страница с рейтингом до проведения боёв



Рейтинговая таблица

Робот	Количество участий	Общее количество очков	Средний балл	Дата изменения
Nemesis	11	23	2,09091	9 апреля 2021 г.
Le Petit Annihilator	7	16	2,28571	9 апреля 2021 г.
Super Joker Mega Kill	8	14	1,75	9 апреля 2021 г.
Killertron	8	13	1,625	9 апреля 2021 г.
Razer	5	10	2,0	9 апреля 2021 г.
Mortis	5	6	1,2	9 апреля 2021 г.
Bigger Brother	2	4	2,0	8 апреля 2021 г.
Dantomkia	2	2	1,0	8 апреля 2021 г.
Sir Killalot	3	0	0,0	9 апреля 2021 г.
Cassius	2	0	0,0	9 апреля 2021 г.
Firestorm	3	0	0,0	8 апреля 2021 г.
Panic Attack	2	0	0,0	8 апреля 2021 г.
Крейсер Варяг	3	0	0,0	8 апреля 2021 г.

Создание турнира



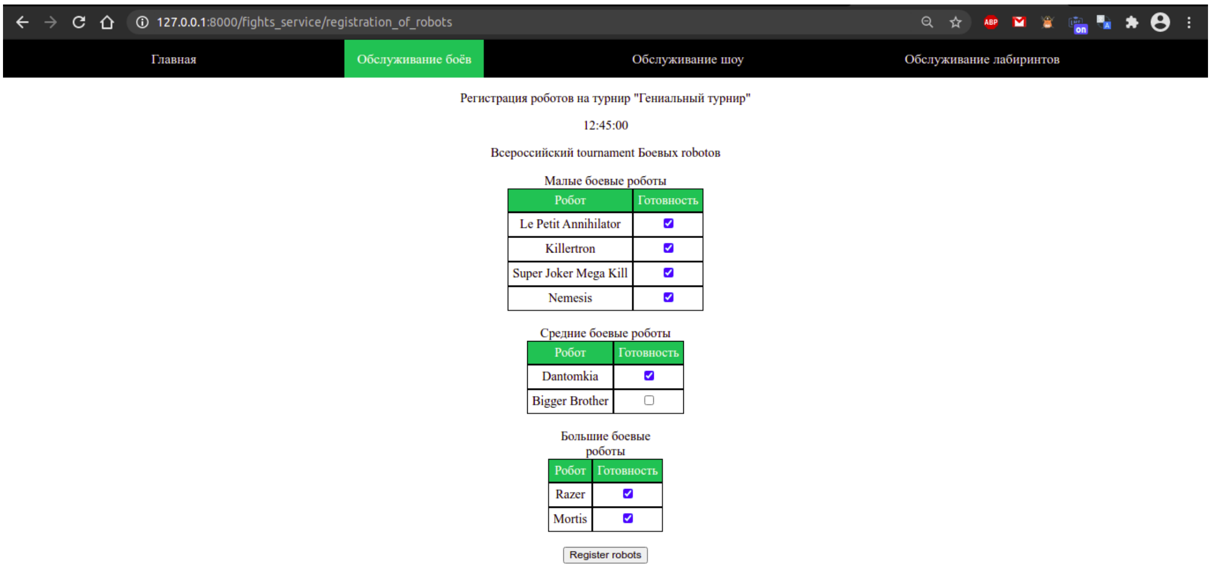
Создание турнира

Title: Гениальный турнир

Time: 12:45:00

Show: Всероссийский tournament Боевых robotов

Регистрация роботов



Регистрация роботов на турнир "Гениальный турнир"

12:45:00

Всероссийский tournament Боевых robotов

Малые боевые роботы

Робот	Готовность
Le Petit Annihilator	☑
Killertron	☑
Super Joker Mega Kill	☑
Nemesis	☑

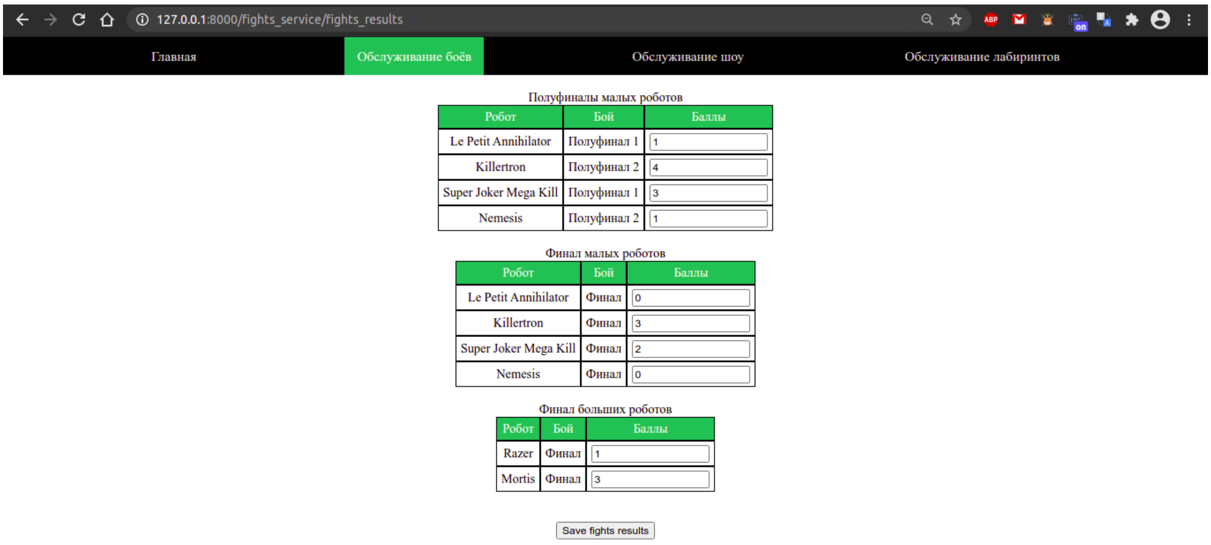
Средние боевые роботы

Робот	Готовность
Dantomkia	☑
Bigger Brother	□

Большие боевые роботы

Робот	Готовность
Razer	☑
Mortis	☑

Запись результатов



Полуфиналы малых роботов

Робот	Бой	Баллы
Le Petit Annihilator	Полуфинал 1	1
Killertron	Полуфинал 2	4
Super Joker Mega Kill	Полуфинал 1	3
Nemesis	Полуфинал 2	1

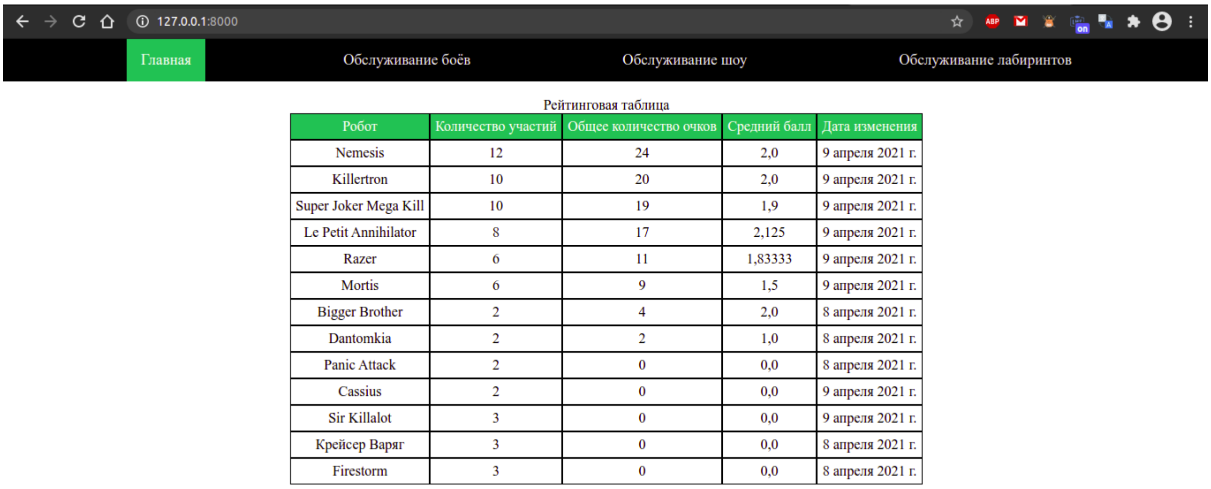
Финал малых роботов

Робот	Бой	Баллы
Le Petit Annihilator	Финал	0
Killertron	Финал	3
Super Joker Mega Kill	Финал	2
Nemesis	Финал	0

Финал больших роботов

Робот	Бой	Баллы
Razer	Финал	1
Mortis	Финал	3

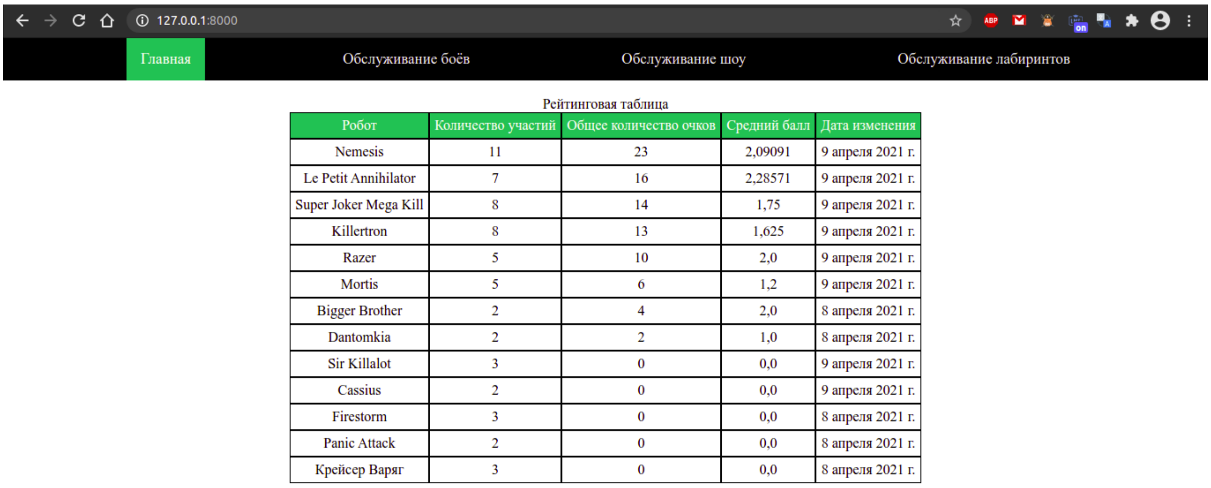
Главная страница после проведения боёв



Рейтинговая таблица

Робот	Количество участий	Общее количество очков	Средний балл	Дата изменения
Nemesis	12	24	2,0	9 апреля 2021 г.
Killertron	10	20	2,0	9 апреля 2021 г.
Super Joker Mega Kill	10	19	1,9	9 апреля 2021 г.
Le Petit Annihilator	8	17	2,125	9 апреля 2021 г.
Razer	6	11	1,83333	9 апреля 2021 г.
Mortis	6	9	1,5	9 апреля 2021 г.
Bigger Brother	2	4	2,0	8 апреля 2021 г.
Dantomkia	2	2	1,0	8 апреля 2021 г.
Panic Attack	2	0	0,0	8 апреля 2021 г.
Cassius	2	0	0,0	9 апреля 2021 г.
Sir Killalot	3	0	0,0	9 апреля 2021 г.
Крейсер Варяг	3	0	0,0	8 апреля 2021 г.
Firestorm	3	0	0,0	8 апреля 2021 г.

Для сравнения приведён снимок главной страницы до проведения боёв



Рейтинговая таблица

Робот	Количество участий	Общее количество очков	Средний балл	Дата изменения
Nemesis	11	23	2,09091	9 апреля 2021 г.
Le Petit Annihilator	7	16	2,28571	9 апреля 2021 г.
Super Joker Mega Kill	8	14	1,75	9 апреля 2021 г.
Killertron	8	13	1,625	9 апреля 2021 г.
Razer	5	10	2,0	9 апреля 2021 г.
Mortis	5	6	1,2	9 апреля 2021 г.
Bigger Brother	2	4	2,0	8 апреля 2021 г.
Dantomkia	2	2	1,0	8 апреля 2021 г.
Sir Killalot	3	0	0,0	9 апреля 2021 г.
Cassius	2	0	0,0	9 апреля 2021 г.
Firestorm	3	0	0,0	8 апреля 2021 г.
Panic Attack	2	0	0,0	8 апреля 2021 г.
Крейсер Варяг	3	0	0,0	8 апреля 2021 г.

Примеры листингов

models.py

```
from django.db import models

class Arena(models.Model):
    title = models.TextField()
    characteristic = models.ForeignKey('ArenaCharacteristic', models.DO_NOTHING, blank=True, null=True)

    def __str__(self):
        return self.title

    class Meta:
        db_table = 'arena'


class ArenaCharacteristic(models.Model):
    seats_count = models.IntegerField(blank=True, null=True)
    address = models.TextField()
    condition = models.TextField()
    technical_inspection_date = models.DateField(blank=True, null=True)
    open_date = models.DateField(blank=True, null=True)
    close_date = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.address

    class Meta:
        db_table = 'arena_characteristic'


class Fight(models.Model):
    tournament = models.ForeignKey('Tournament', models.DO_NOTHING, blank=True, null=True)
    phase = models.TextField(blank=True, null=True)
    class_field = models.ForeignKey('RobotClasses', models.DO_NOTHING, db_column='class_id', blank=True, null=True)  # Field renamed because it was a Python reserved word.

    def __str__(self):
        return self.phase

    class Meta:
        db_table = 'fight'
```

```
class FightParticipation(models.Model):
    fight = models.OneToOneField(Fight, models.DO_NOTHING, primary_key=True)
    robot = models.ForeignKey('Robot', models.DO_NOTHING)
    scores = models.IntegerField(blank=True, null=True)
    comment = models.TextField(blank=True, null=True)

    class Meta:
        db_table = 'fight_participation'
        unique_together = (('fight', 'robot'),)


class Member(models.Model):
    surname = models.TextField(blank=True, null=True)
    name = models.TextField()
    team = models.ForeignKey('Team', models.DO_NOTHING, blank=True, null=True)
    role = models.TextField(blank=True, null=True)
    entrance_date = models.DateField()
    exit = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.name + ' ' + self.surname

    class Meta:
        db_table = 'member'


class Race(models.Model):
    tournament = models.ForeignKey('Tournament', models.DO_NOTHING, blank=True,
null=True)
    robot = models.ForeignKey('Robot', models.DO_NOTHING, blank=True, null=True)
    race_time = models.TimeField(blank=True, null=True)

    class Meta:
        db_table = 'race'


class RatingTable(models.Model):
    robot = models.ForeignKey('Robot', models.DO_NOTHING, blank=True, null=True)
    participate_count = models.IntegerField(blank=True, null=True)
    scores = models.IntegerField(blank=True, null=True)
    average_score = models.FloatField(blank=True, null=True)
    last_modification_date = models.DateField(blank=True, null=True)

    class Meta:
        db_table = 'rating_table'


class Robot(models.Model):
    title = models.TextField(unique=True)
    team = models.ForeignKey('Team', models.DO_NOTHING, blank=True, null=True)
    class_field = models.ForeignKey('RobotClasses', models.DO_NOTHING,
db_column='class_id', blank=True, null=True)  # Field renamed because it was a Python
reserved word.
    first_participation_date = models.DateField(blank=True, null=True)
    last_participation_date = models.DateField(blank=True, null=True)
    condition = models.BooleanField(blank=True, null=True)

    def __str__(self):
          return self.title

    class Meta:
        db_table = 'robot'


class RobotClasses(models.Model):
    size = models.TextField()  # This field type is a guess.
    drone_control = models.BooleanField(blank=True, null=True)
    role = models.TextField()  # This field type is a guess.

    def __str__(self):
        return self.size + ' ' + self.role

    class Meta:
        db_table = 'robot_classes'


class Show(models.Model):
    title = models.TextField()
    fromat = models.TextField(blank=True, null=True)
    id_arena = models.ForeignKey(Arena, models.DO_NOTHING, db_column='id_arena',
blank=True, null=True)
    show_date = models.DateField()
    show_time = models.TimeField(blank=True, null=True)

    def __str__(self):
        return self.title

    class Meta:
        db_table = 'show'


class ShowParticipation(models.Model):
    show = models.OneToOneField(Show, models.DO_NOTHING, primary_key=True)
    robot = models.ForeignKey(Robot, models.DO_NOTHING)
    comment = models.TextField(blank=True, null=True)

    class Meta:
        db_table = 'show_participation'
        unique_together = (('show', 'robot'),)
```

```
class Team(models.Model):
    title = models.TextField()
    creation_date = models.DateField()
    close_date = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.title

    class Meta:
        db_table = 'team'


class Tournament(models.Model):
    title = models.TextField()
    show = models.ForeignKey(Show, models.DO_NOTHING, blank=True, null=True)
    tournament_time = models.TimeField(blank=True, null=True)

    def __str__(self):
        return self.title

    class Meta:
        db_table = 'tournament'
```

urls.py

```
from django.urls import path

from . import views

app_name = 'robot_reg'

urlpatterns = [
          path('', views.index, name = 'index'),
          path('fights_service', views.tournament_create, name = 'tournament_create'),
          path('fights_service/registration_of_robots', views.registration_of_robots, name
= 'registration_of_robots'),
          path('fights_service/fights_results', views.fights_results, name = 'fights_results'),
          path('race_service', views.race_tournament_create, name =
'race_tournament_create'),
          path('race_service/registration_of_robots', views.race_registration_of_robots,
name = 'race_registration_of_robots'),
          path('race_service/race_results', views.race_results, name = 'race_results'),
          path('show_service', views.select_robots_to_show, name =
'select_robots_to_show')
]
```

views.py

```
from django.shortcuts import render

from django.db import connection

from collections import defaultdict

from .models import Robot
from .models import Arena
from .models import ArenaCharacteristic
from .models import Fight
from .models import FightParticipation
from .models import Member
from .models import Race
from .models import RatingTable
from .models import RobotClasses
from .models import Show
from .models import ShowParticipation
from .models import Team
from .models import Tournament

from django.http import Http404, HttpResponseRedirect

from .forms import TournamentCreateForm

def index(request):
	if request.method == "POST":
		request_data = dict(request.POST)
		del request_data['csrfmiddlewaretoken']
		referer = request_data['referer'][0]
		del request_data['referer']

		cursor = connection.cursor()
		results_to_write_list = list()

		if referer == "fight_service":
			class Result_of_robot_fight:
				robot_id = int()
				fight_id = int()
				score = int()

			for key in request_data:
				space_pos = key.find(' ')
				result_of_robot_fight = Result_of_robot_fight()
				result_of_robot_fight.robot_id = int(key[:space_pos])
				result_of_robot_fight.fight_id = int(key[space_pos:])
				result_of_robot_fight.score = int(request_data[key][0])
				if result_of_robot_fight.score > 0:
					results_to_write_list.append(result_of_robot_fight)
			for result in results_to_write_list:
				cursor.execute("select write_fight_result_en(%s, %s, %s,
%s)", [result.fight_id, result.robot_id, result.score, ''])
		if referer == "race_service":
			class Result_of_robot_race:
				robot_id = int()
				result_time = ''

			tournament_id = request_data['tournament_id'][0]
			del request_data['tournament_id']

			for key in request_data:
				result_of_robot_race = Result_of_robot_race()
				result_of_robot_race.robot_id = key
				result_of_robot_race.result_time = request_data[key][0]
				results_to_write_list.append(result_of_robot_race)

			for result in results_to_write_list:
				cursor.execute("select race_result_en(%s, %s, %s)",
[tournament_id, result.robot_id, result.result_time])
		if referer == "show_service":
			show_id = int()
			show_id = request_data['show'][0]
			del request_data['show']

			for key in request_data:
				results_to_write_list.append(key)

			for result in results_to_write_list:
				cursor.execute("select show_participation_en(%s, %s,
%s)", [show_id, result, ''])

		cursor.execute("select robot_default_condition_en()")

	robots_rating_list = RatingTable.objects.order_by('-scores')
	return render(request, 'main/hello_page.html', {'robots_rating_list':
robots_rating_list})

#
#Обслуживание боёв
#
def tournament_create(request):
	tournament_create_form = TournamentCreateForm()
	return render(request, 'fights_service/tournament_creations.html',
{"tournament_create_form": tournament_create_form})


def registration_of_robots(request):
	if request.method == "POST":
        new_tournament = Tournament()
        new_tournament.title = request.POST.get("title")
        new_tournament.tournament_time = request.POST.get("time")
        new_tournament.show = Show.objects.get(id =
request.POST.get("show"))
        new_tournament.save()

        robots_list = Robot.objects.all()

        small_class = RobotClasses.objects.get(id = 1)
        small_robots_list = Robot.objects.filter(class_field = small_class)

        medium_class = RobotClasses.objects.get(id = 5)
        medium_robots_list = Robot.objects.filter(class_field = medium_class)

        big_class = RobotClasses.objects.get(id = 8)
        big_robots_list = Robot.objects.filter(class_field = big_class)

    return render(request, 'fights_service/registration_robots.html',
{"small_robots_list": small_robots_list, "medium_robots_list": medium_robots_list,
"big_robots_list": big_robots_list, "tournament": new_tournament})


def fights_results(request):
    if request.method == "POST":
        data_from_post_robot_ids = list(dict(request.POST).keys())
        data_from_post_robot_ids.remove('csrfmiddlewaretoken')
        data_from_post_robot_ids.remove('tournament_id')

        robot_ready_ids = [int(key) for key in data_from_post_robot_ids]
        ready_robots = Robot.objects.filter(id__in = robot_ready_ids)

        for robot in ready_robots:
            robot.condition = True
            robot.save()

        small_class = RobotClasses.objects.get(id = 1)
        medium_class = RobotClasses.objects.get(id = 5)
        big_class = RobotClasses.objects.get(id = 8)

        tournament_id = request.POST.get("tournament_id")
        tournament = Tournament.objects.get(id = tournament_id)

        cursor = connection.cursor()
        cursor.execute("select organize_figths_en(%s)", [tournament_id])


    #Создание соотношений боёв к роботам
        tournament_fights = Fight.objects.filter(tournament = tournament_id)
    #Определение элемента спика
        class Fight_of_robot:
            robot_id = int()
            fight_id = int()
            fight_class = int()
            phase = int()

    #сам список
        fights_of_robots_list = list()

    #формирование списка
    #заполняем сетку малых
        small_robots_count = Robot.objects.filter(id__in = robot_ready_ids,
class_field = 1).count()

        small_final_exist = False
        if small_robots_count > 1:
            small_final_exist = True
        #список малых на финал
            small_final_robots = Robot.objects.filter(id__in = robot_ready_ids,
class_field = small_class)
        #малые полуфиналы
            small_final = Fight.objects.get(tournament = tournament_id,
class_field = 1, phase = "Финал")
        #запись роботов в финал
            for robot in ready_robots:
                if robot in small_final_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = small_final.id
                    fight_of_robot.fight_class = 1
                    fight_of_robot.phase = 2
                    fights_of_robots_list.append(fight_of_robot)


        small_semifinals_exist = False
        if small_robots_count > 3:
            small_semifinals_exist = True
        #списки малых роботов на полуфиналы
            small_semifinal_1_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = small_class)[::2]
            small_semifinal_2_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = small_class)[1::2]
        #малые полуфиналы
            small_semifinal_1 = Fight.objects.get(tournament =
tournament_id, class_field = 1, phase = "Полуфинал 1")
            small_semifinal_2 = Fight.objects.get(tournament =
tournament_id, class_field = 1, phase = "Полуфинал 2")
        #распределение роботов по полуфиналам
            for robot in ready_robots:
                if robot in small_semifinal_1_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = small_semifinal_1.id
                    fight_of_robot.fight_class = 1
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)
                if robot in small_semifinal_2_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = small_semifinal_2.id
                    fight_of_robot.fight_class = 1
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)

    #заполняем сетку средних
        medium_robots_count = Robot.objects.filter(id__in = robot_ready_ids,
class_field = 5).count()

        medium_final_exist = False
        if medium_robots_count > 1:
            medium_final_exist = True
        #список средних на финал
            medium_final_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = medium_class)
        #малые полуфиналы
            medium_final = Fight.objects.get(tournament = tournament_id,
class_field = 5, phase = "Финал")
        #запись роботов в финал
            for robot in ready_robots:
                if robot in medium_final_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = medium_final.id
                    fight_of_robot.fight_class = 2
                    fight_of_robot.phase = 2
                    fights_of_robots_list.append(fight_of_robot)



        medium_semifinals_exist = False
        if medium_robots_count > 3:
            medium_semifinals_exist = True
        #списки средних роботов на полуфиналы
            medium_semifinal_1_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = medium_class)[::2]
            medium_semifinal_2_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = medium_class)[1::2]
        #малые полуфиналы
            medium_semifinal_1 = Fight.objects.get(tournament =
tournament_id, class_field = 5, phase = "Полуфинал 1")
```

```
            medium_semifinal_2 = Fight.objects.get(tournament =
tournament_id, class_field = 5, phase = "Полуфинал 2")
        #распределение роботов по полуфиналам
            for robot in ready_robots:
                if robot in medium_semifinal_1_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = medium_semifinal_1.id
                    fight_of_robot.fight_class = 2
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)
                if robot in medium_semifinal_2_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = medium_semifinal_2.id
                    fight_of_robot.fight_class = 2
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)

    #заполняем сетку больших
        big_robots_count = Robot.objects.filter(id__in = robot_ready_ids,
class_field = 8).count()

        big_final_exist = False
        if big_robots_count > 1:
            big_final_exist = True
        #список больших на финал
            big_final_robots = Robot.objects.filter(id__in = robot_ready_ids,
class_field = big_class)
        #малые полуфиналы
            big_final = Fight.objects.get(tournament = tournament_id,
class_field = 8, phase = "Финал")
        #запись роботов в финал
            for robot in ready_robots:
                if robot in big_final_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = big_final.id
                    fight_of_robot.fight_class = 3
                    fight_of_robot.phase = 2
                    fights_of_robots_list.append(fight_of_robot)



        big_semifinals_exist = False
        if big_robots_count > 3:
            big_semifinals_exist = True
        #списки больших роботов на полуфиналы
            big_semifinal_1_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = big_class)[::2]
```

```
            big_semifinal_2_robots = Robot.objects.filter(id__in =
robot_ready_ids, class_field = big_class)[1::2]
        #малые полуфиналы
            big_semifinal_1 = Fight.objects.get(tournament = tournament_id,
class_field = 8, phase = "Полуфинал 1")
            big_semifinal_2 = Fight.objects.get(tournament = tournament_id,
class_field = 8, phase = "Полуфинал 2")
        #распределение роботов по полуфиналам
            for robot in ready_robots:
                if robot in big_semifinal_1_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = big_semifinal_1.id
                    fight_of_robot.fight_class = 3
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)
                if robot in big_semifinal_2_robots:
                    fight_of_robot = Fight_of_robot()
                    fight_of_robot.robot_id = robot.id
                    fight_of_robot.fight_id = big_semifinal_2.id
                    fight_of_robot.fight_class = 3
                    fight_of_robot.phase = 1
                    fights_of_robots_list.append(fight_of_robot)


    return render(request, 'fights_service/confirm_fights_results.html',
{"ready_robots": ready_robots, "small_class": small_class, "medium_class":
medium_class, "big_class": big_class, "tournament_fights": tournament_fights,
"fights_of_robots_list": fights_of_robots_list, "small_semifinals_exist":
small_semifinals_exist, "small_final_exist": small_final_exist,
"medium_semifinals_exist": medium_semifinals_exist, "medium_final_exist":
medium_final_exist, "big_semifinals_exist": big_semifinals_exist, "big_final_exist":
big_final_exist})


#
#Обслуживание забегов
#
def race_tournament_create(request):
    tournament_create_form = TournamentCreateForm()
    return render(request, 'race_service/race_tournament_creation.html',
{"tournament_create_form": tournament_create_form})

def race_registration_of_robots(request):
    if request.method == "POST":
        new_tournament = Tournament()
        new_tournament.title = request.POST.get("title")
        new_tournament.tournament_time = request.POST.get("time")
        new_tournament.show = Show.objects.get(id =
request.POST.get("show"))
```

```
        new_tournament.save()

        race_class = RobotClasses.objects.get(id = 3)
        race_robots_list = Robot.objects.filter(class_field = race_class)

    return render(request, 'race_service/registration_robots.html', {"tournament":
new_tournament, "race_robots_list": race_robots_list})

def race_results(request):
    if request.method == "POST":
        data_from_post_robot_ids = list(dict(request.POST).keys())
        data_from_post_robot_ids.remove('csrfmiddlewaretoken')
        data_from_post_robot_ids.remove('tournament_id')

        robot_ready_ids = [int(key) for key in data_from_post_robot_ids]
        ready_robots = Robot.objects.filter(id__in = robot_ready_ids)

        for robot in ready_robots:
            robot.condition = True
            robot.save()

        race_class = RobotClasses.objects.get(id = 3)

        tournament_id = request.POST.get("tournament_id")
        tournament = Tournament.objects.get(id = tournament_id)

    return render(request, 'race_service/confirm_race_results.html', {"tournament":
tournament, "ready_robots": ready_robots})


#
#Обслуживание шоу
#
def select_robots_to_show(request):
    show_classes = RobotClasses.objects.filter(role = "шоу")
    show_robots_list = Robot.objects.filter(class_field__in = show_classes)

    shows = Show.objects.all()

    shows_size = Show.objects.all().count()

    return render(request, 'show_service/register_robots_to_show.html',
{"show_robots_list": show_robots_list, "shows": shows, "shows_size": shows_size})
```

hello_page.html шаблон

```
{% extends 'base.html' %}

{% block title %} Robofight service {% endblock %}

{% block content %}

           <div class="navbar">
    <a class="active">Главная</a>
    <a href="{% url 'robot_reg:tournament_create' %}">Обслуживание боёв</a>
    <a href="{% url 'robot_reg:select_robots_to_show' %}">Обслуживание шоу</a>
    <a href="{% url 'robot_reg:race_tournament_create' %}">Обслуживание лабиринтов</a>
  </div>
  <div class="under_navbar">
    <a>Главная</a>
    <a>Регистрация робота</a>
    <a>Обслуживание боёв</a>
    <a>Обслуживание шоу</a>
    <a>Обслуживание лабиринтов</a>
  </div>

  <div class="container">
    <div class="content_item">
      <br>
      {% if robots_rating_list %}
        <table>
          <caption>Рейтинговая таблица</caption>
          <thead>
            <tr>
              <td>Робот</td>
              <td>Количество участий</td>
              <td>Общее количество очков</td>
              <td>Средний балл</td>
              <td>Дата изменения</td>
            </tr>
          </thead>
        {% for a in robots_rating_list %}
          <tr>
            <td>{{a.robot}}</td>
            <td>{{a.participate_count}}</td>
            <td>{{a.scores}}</td>
            <td>{{a.average_score}}</td>
            <td>{{a.last_modification_date}}</td>
          </tr>
        {% endfor %}

        </table>
      {% else %}
        В базе нет рейтинга роботов
      {% endif %}
    </div>
  </div>

{% endblock %}
```