Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по зачетной работе

Тема: Приложение для отслеживания задач. Таск-трекер

Дисциплина: Языки Интернет-программирования

Студент	ИУ6-31 Б	21.12.2022	И.А. Абызов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		21.12.2022	Д.В. Малахов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022

Цель работы

Цель работы — продемонстрировать практические навыки создания веб-приложения с использованием средств Ruby on Rails, полученные в рамках курса «Языки Интернет-программирования».

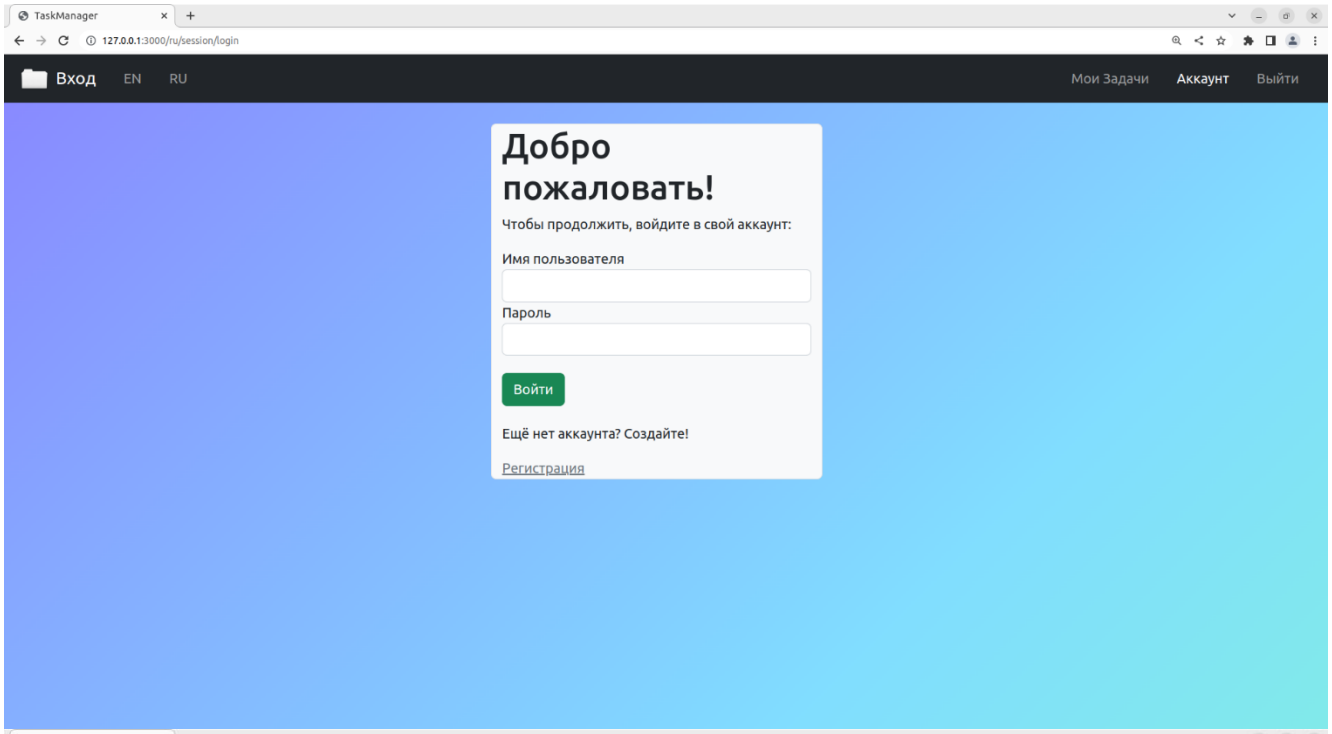
Тема зачетной работы — создание веб-приложения для отслеживания своих задач, как личных, так и относящихся к какой-либо группе людей.

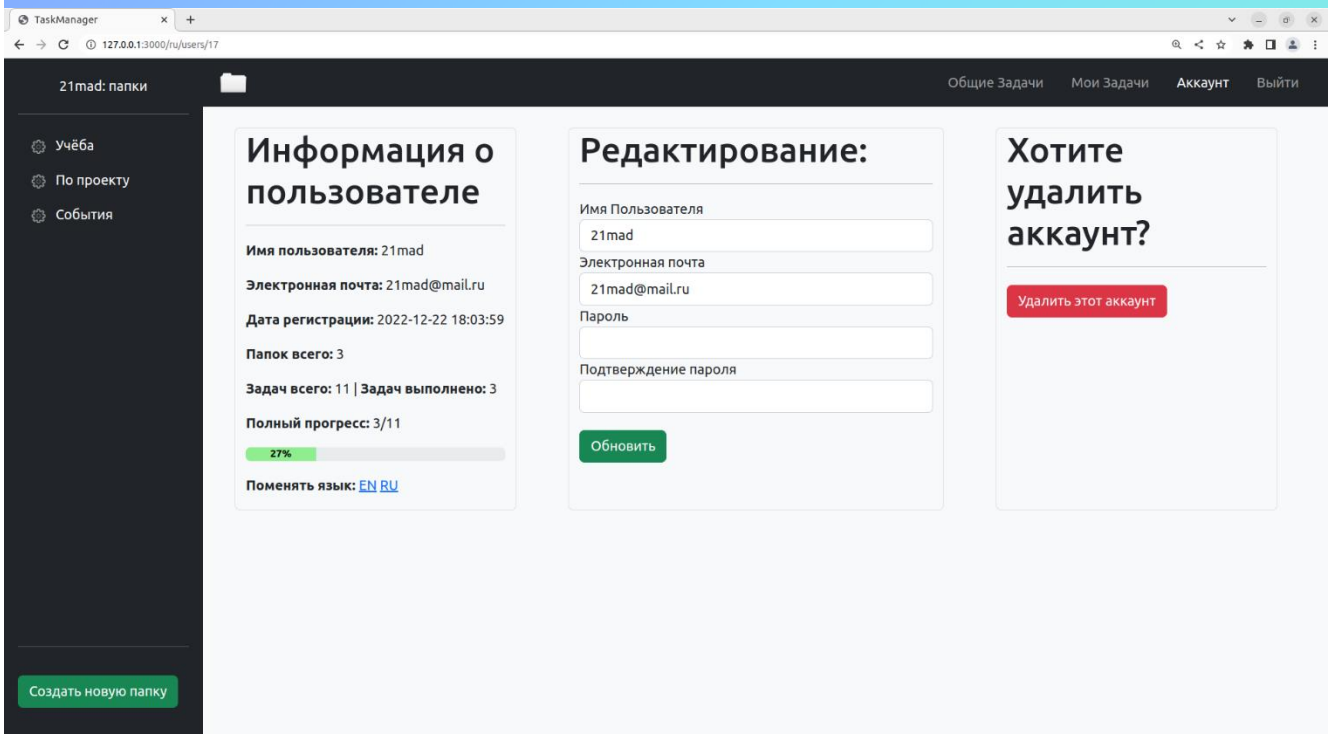
Функционал моего веб-приложения:

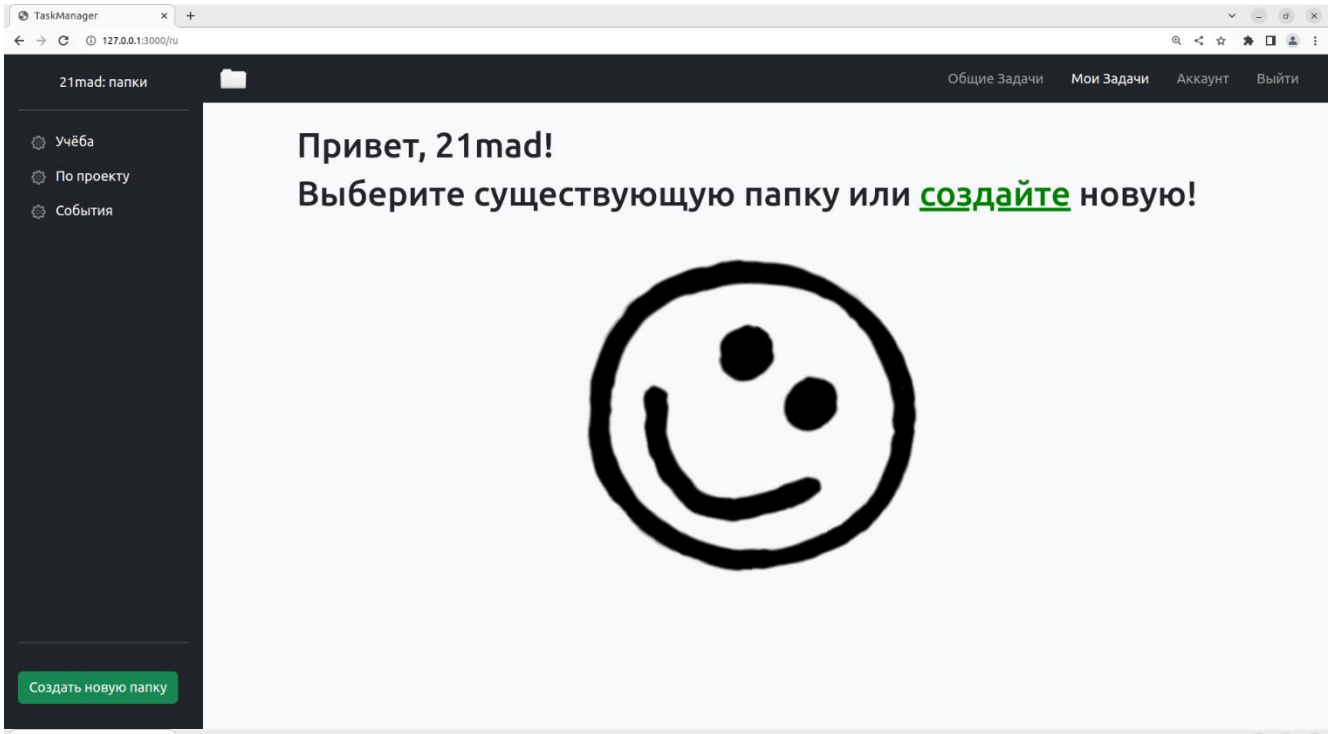
- Для многопользовательской работы и работы в приложении в целом созданы: модель пользователя, его регистрация и аутентификация, контроль сессий.
- Для личных задач – страница «Мои Задачи», в которой пользователи могут создавать свои папки для задач разной тематики. Папки можно редактировать, изменять настройки цветовых маркеров, удалять. Задания делятся на две секции: выполненные и «в процессе», которые с свою очередь выделяются цветовыми маркерами в зависимости от дедлайна.
- Для общих задач – страница «Общие Задачи», в которой пользователи так же могут создавать свои папки, но уже с настраиваемым доступом. В свою папку можно добавлять и удалять участников по их никнейму. Редактирование папки доступно только для владельца. В разделе задач приводится общая статистика выполнения, а также статистика для каждого участника и список последних выполненных задач. Задачи помечаются ником пользователя после её выполнения.
- Для работы с пользователями в приложении была создана роль администратора, которая открывает доступ ко списку всех пользователей, просмотру и редактированию их аккаунтов.
- Интернационализация. Возможность выбора языка: английского и русского.

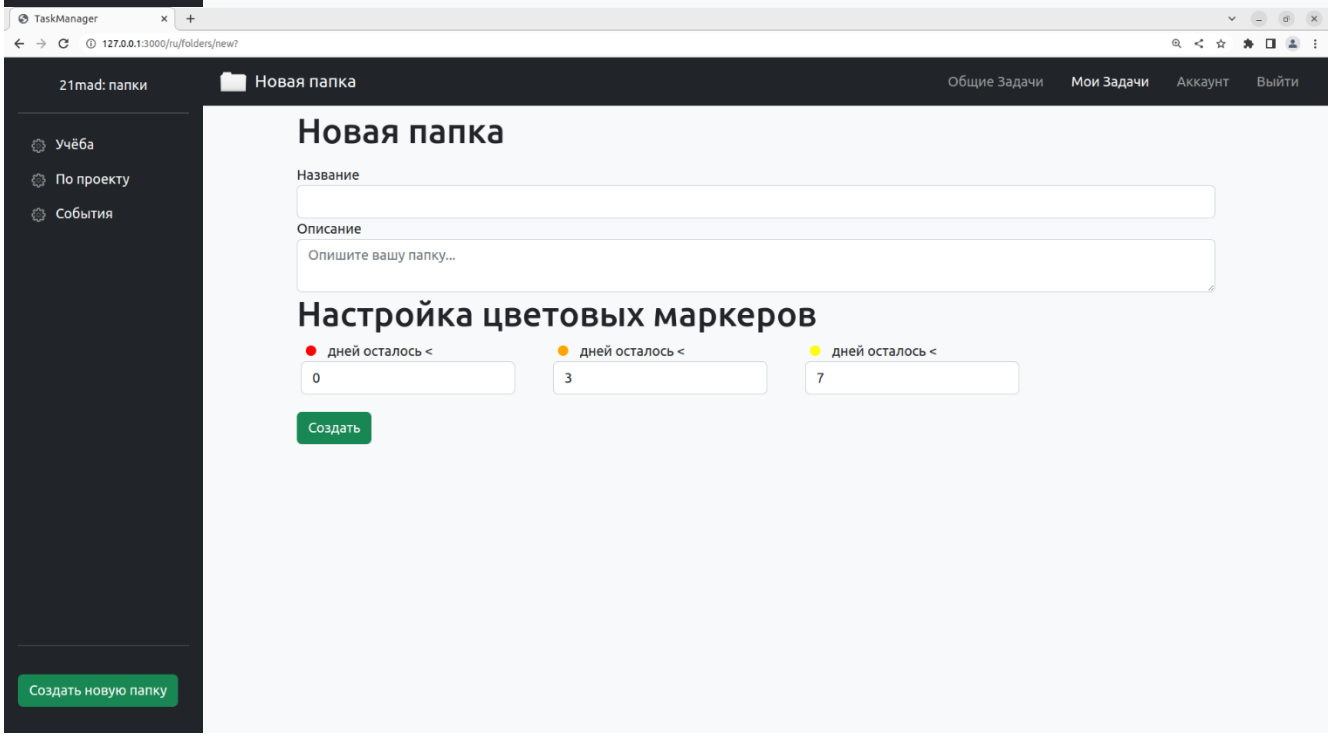
Исходный код приложения можно так же посмотреть на GitHub: https://github.com/21mad/TaskManager

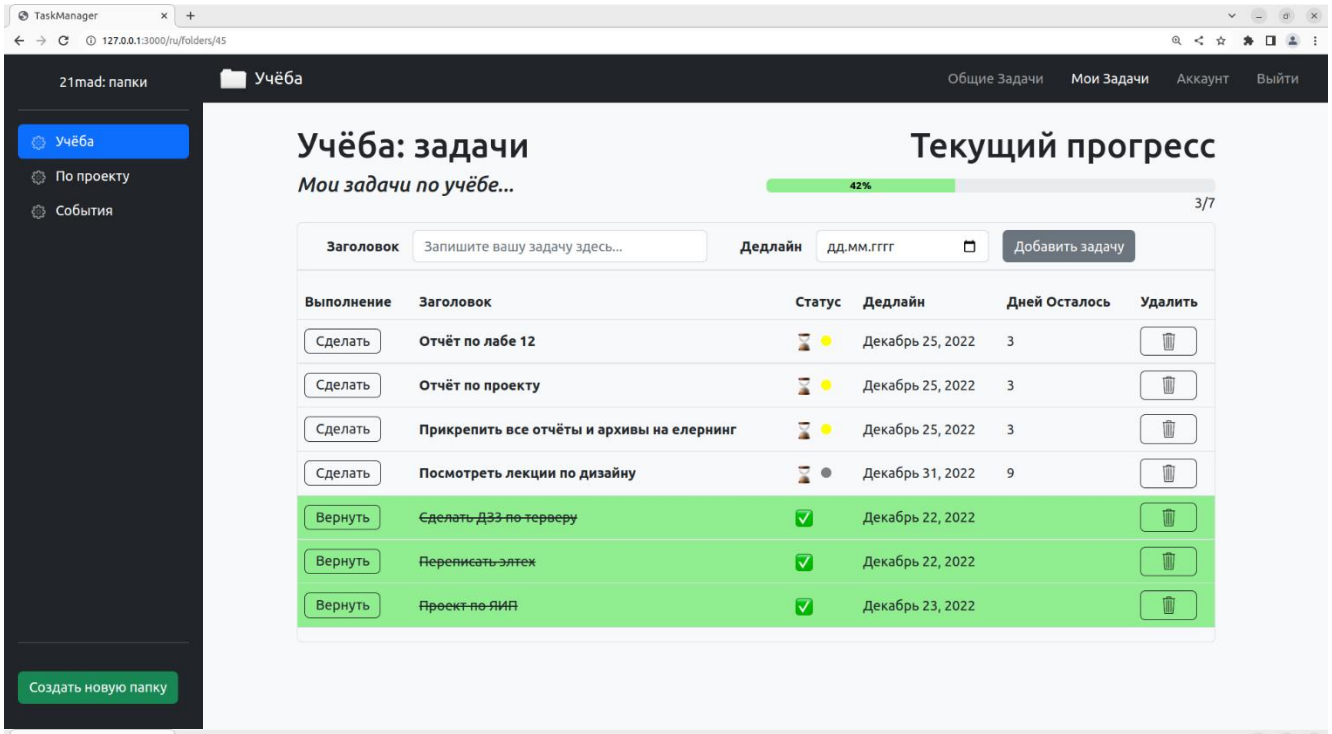
Работа приложения

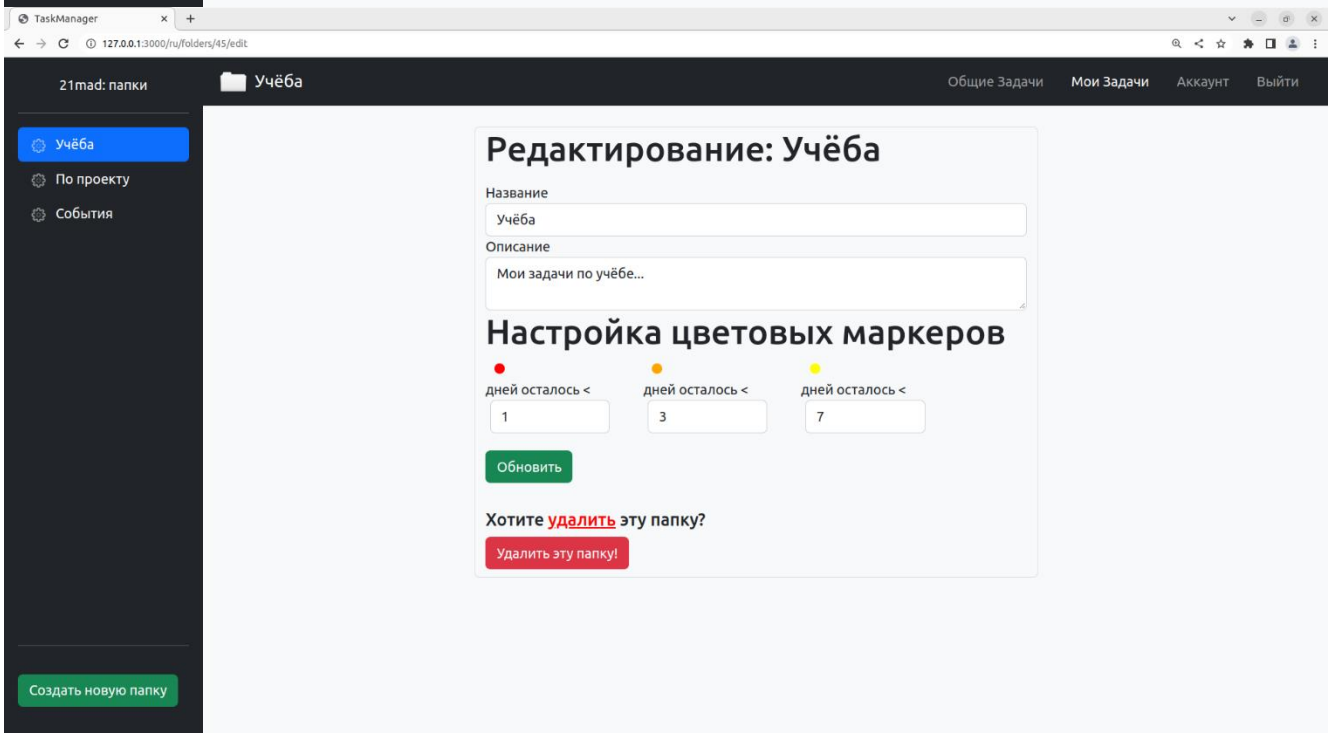


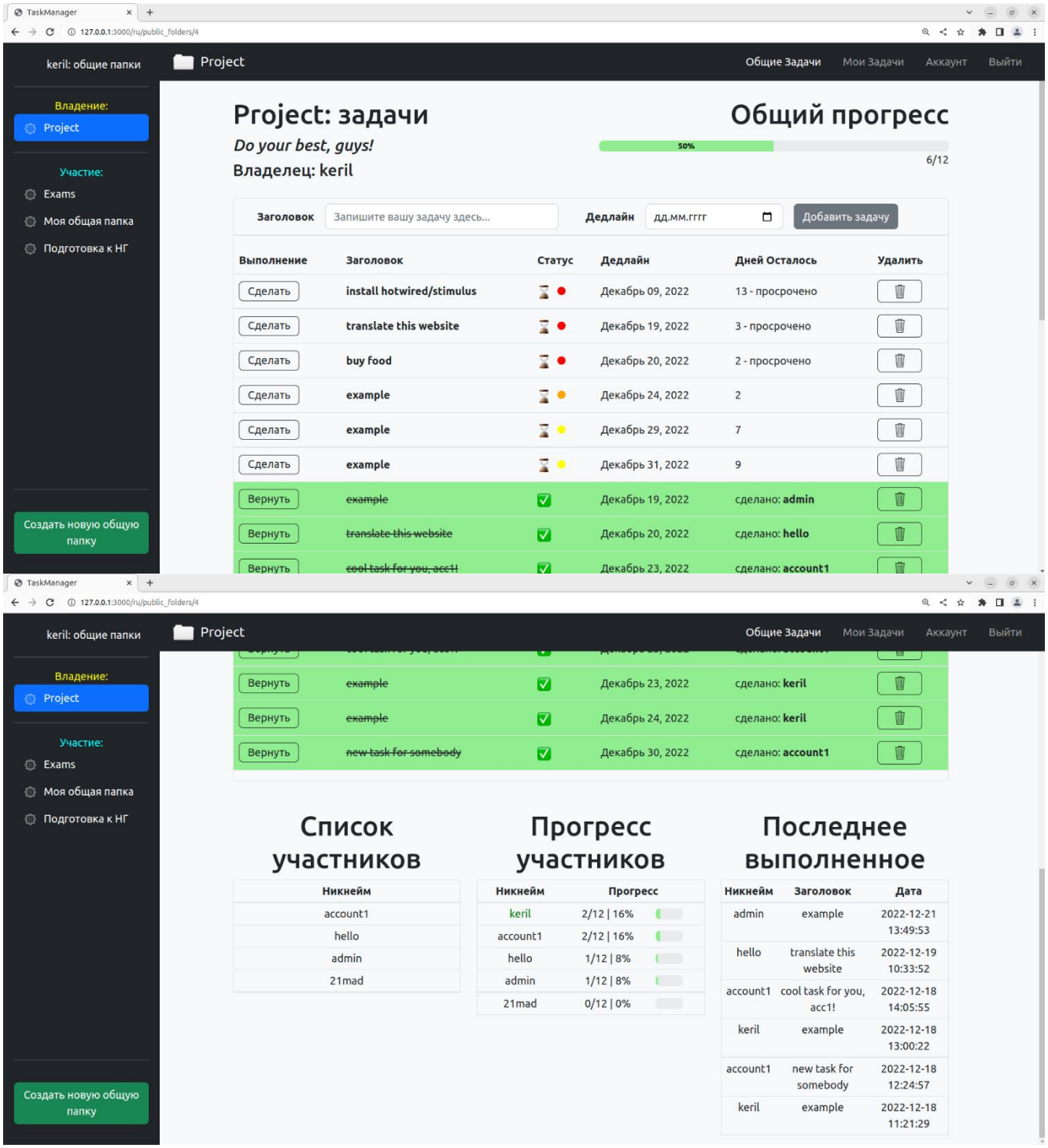



TaskManager
127.0.0.1:3000/ru
21mad: папки
Учёба
По проекту
События
Создать новую папку
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Привет, 21mad!
Выберите существующую папку или создайте новую!

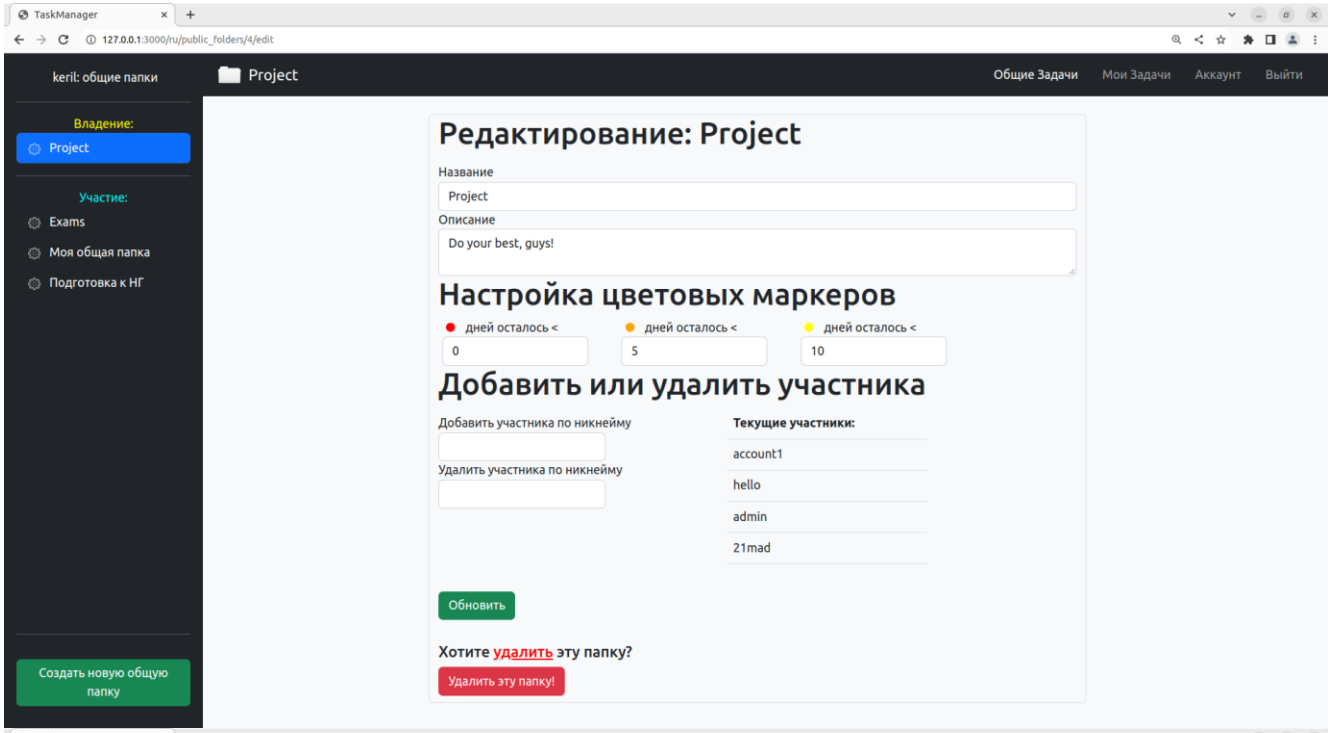

TaskManager
127.0.0.1:3000/ru/folders/new?
21mad: папки
Учёба
По проекту
События
Создать новую папку
Новая папка
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Новая папка
Название
Описание
Опишите вашу папку...
Настройка цветовых маркеров
дней осталось <
0
дней осталось <
3
дней осталось <
7
Создать

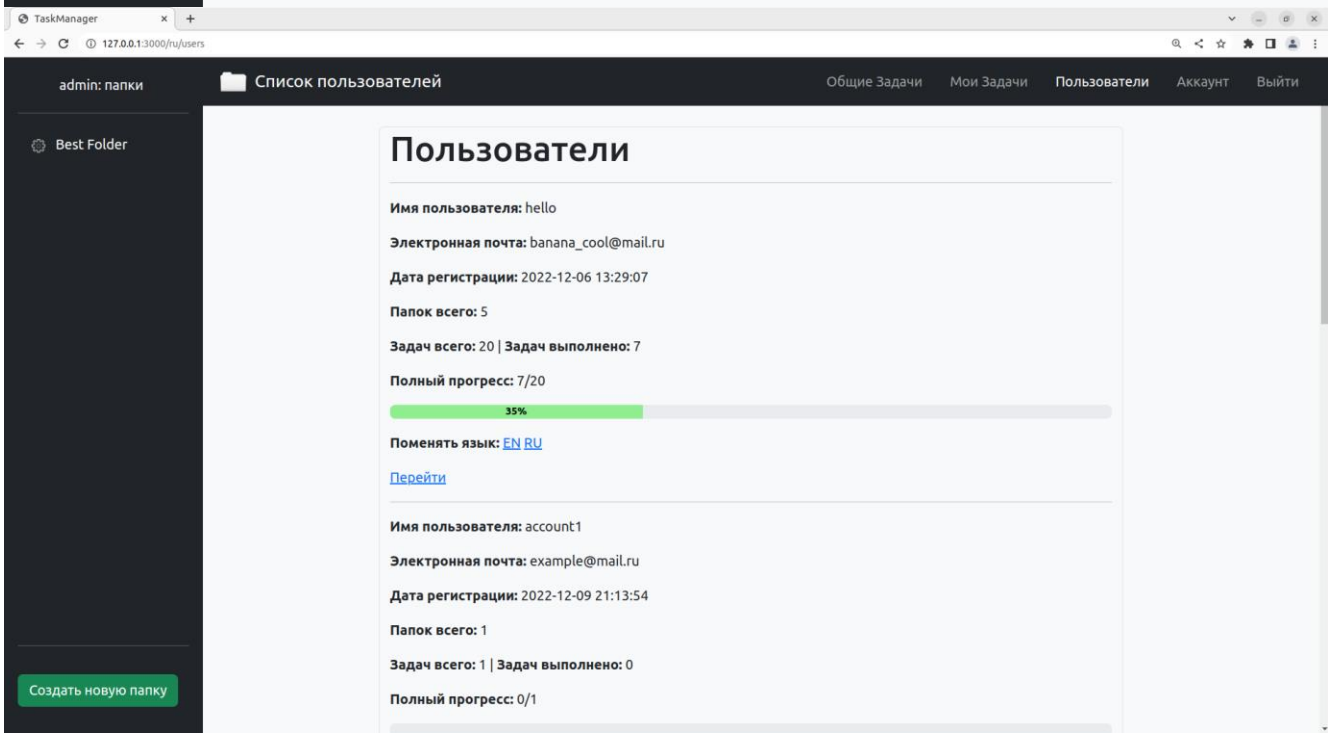

TaskManager
127.0.0.1:3000/ru/folders/45
21mad: папки
Учёба
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Учёба
По проекту
События
Учёба: задачи
Мои задачи по учёбе...
Текущий прогресс
42%
3/7
Заголовок
Запишите вашу задачу здесь...
Дедлайн
дд.мм.гггг
Добавить задачу
Выполнение
Заголовок
Статус
Дедлайн
Дней Осталось
Удалить
Сделать
Отчёт по лабе 12
Декабрь 25, 2022
3
Сделать
Отчёт по проекту
Декабрь 25, 2022
3
Сделать
Прикрепить все отчёты и архивы на елернинг
Декабрь 25, 2022
3
Сделать
Посмотреть лекции по дизайну
Декабрь 31, 2022
9
Вернуть
Сделать ДЗ3 по терверу
Декабрь 22, 2022
Вернуть
Переписать эятех
Декабрь 22, 2022
Вернуть
Проект по ЯИП
Декабрь 23, 2022
Создать новую папку


TaskManager
127.0.0.1:3000/ru/folders/45/edit
21mad: папки
Учёба
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Учёба
По проекту
События
Редактирование: Учёба
Название
Учёба
Описание
Мои задачи по учёбе...
Настройка цветовых маркеров
дней осталось <
1
дней осталось <
3
дней осталось <
7
Обновить
Хотите удалить эту папку?
Удалить эту папку!
Создать новую папку


TaskManager
127.0.0.1:3000/ru/public_folders/4
keril: общие папки
Project
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Владение:
Project
Участие:
Exams
Моя общая папка
Подготовка к НГ
Создать новую общую папку
Project: задачи
Do your best, guys!
Владелец: keril
Общий прогресс
50%
6/12
Заголовок
Запишите вашу задачу здесь...
Дедлайн
дд.мм.гггг
Добавить задачу
Выполнение
Заголовок
Статус
Дедлайн
Дней Осталось
Удалить
Сделать
install hotwired/stimulus
Декабрь 09, 2022
13 - просрочено
Сделать
translate this website
Декабрь 19, 2022
3 - просрочено
Сделать
buy food
Декабрь 20, 2022
2 - просрочено
Сделать
example
Декабрь 24, 2022
2
Сделать
example
Декабрь 29, 2022
7
Сделать
example
Декабрь 31, 2022
9
Вернуть
example
Декабрь 19, 2022
сделано: admin
Вернуть
translate this website
Декабрь 20, 2022
сделано: hello
Вернуть
cool task for you, acc1!
Декабрь 23, 2022
сделано: account1

TaskManager
127.0.0.1:3000/ru/public_folders/4
keril: общие папки
Project
Общие Задачи
Мои Задачи
Аккаунт
Выйти
Владение:
Project
Участие:
Exams
Моя общая папка
Подготовка к НГ
Создать новую общую папку
Вернуть
example
Декабрь 23, 2022
сделано: keril
Вернуть
example
Декабрь 24, 2022
сделано: keril
Вернуть
new task for somebody
Декабрь 30, 2022
сделано: account1
Список участников
Никнейм
account1
hello
admin
21mad
Прогресс участников
Никнейм
Прогресс
keril
2/12 | 16%
account1
2/12 | 16%
hello
1/12 | 8%
admin
1/12 | 8%
21mad
0/12 | 0%
Последнее выполненное
Никнейм
Заголовок
Дата
admin
example
2022-12-21 13:49:53
hello
translate this website
2022-12-19 10:33:52
account1
cool task for you, acc1!
2022-12-18 14:05:55
keril
example
2022-12-18 13:00:22
account1
new task for somebody
2022-12-18 12:24:57
keril
example
2022-12-18 11:21:29





Измененные и добавленные файлы бизнес-логики приложения

Модели

Текст файла user.rb

```
class User < ApplicationRecord
    has_secure_password
    validates_uniqueness_of :username
    validates_presence_of :username
end
class User < ApplicationRecord
```

```
  has_secure_password
  validates_uniqueness_of :username
  validates_presence_of :username
  validates_presence_of :password
  validates_confirmation_of :password
  validates_uniqueness_of :email
  validates_presence_of :email
  validates :email, format: { with: URI::MailTo::EMAIL_REGEXP, message:
I18n.t("is_invalid") } # email validation
  validates :username, format: { with: /\A(?=.{5,})/x, message: I18n.t("is_short") }

  PASSWORD_FORMAT = /\A
  (?=.{8,})          # Must contain 8 or more characters
  (?=.*\d)           # Must contain a digit
  (?=.*[a-z])        # Must contain a lower case character
  (?=.*[A-Z])        # Must contain an upper case character
  (?=.*[[:^alnum:]]) # Must contain a symbol
  /x
  # validates :password, format: { with: PASSWORD_FORMAT, message: "is not
secure. Must contain 8 or more characters" }
  # it works, but I'm too lazy to come up with passwords...

  has_many :folders, dependent: :destroy
end
```

Текст файла folder.rb

```
class Folder < ApplicationRecord
  belongs_to :user
  has_many :tasks, dependent: :destroy

  validates :name, presence: true
  validates :name, uniqueness: { scope: [:user_id] }
end
```

Текст файла task.rb

```
class Task < ApplicationRecord
  belongs_to :folder, optional: true # добаить миграцию на отмену null: false если
будет ругаться
  belongs_to :public_folder, optional: true

  validates :title, presence: true
  validates :deadline, presence: true
end
```

Текст файла public_folder.rb

```
class PublicFolder < ApplicationRecord
```

```
  belongs_to :user
  has_many :tasks, dependent: :destroy

  validates :name, presence: true
  validates :name, uniqueness: { scope: [:user_id] }
end
```

Текст файла создания БД schema.rb

```
ActiveRecord::Schema[7.0].define(version: 2022_12_18_095536) do
  create_table "folders", force: :cascade do |t|
    t.string "name"
    t.integer "user_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "description"
    t.text "colors", default: "{\"red\":0,\"orange\":3,\"yellow\":7}"
    t.index ["user_id"], name: "index_folders_on_user_id"
  end

  create_table "public_folders", force: :cascade do |t|
    t.string "name"
    t.string "description"
    t.text "colors", default: "{\"red\":0,\"orange\":3,\"yellow\":7}"
    t.text "members", default: "[]"
    t.integer "user_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_public_folders_on_user_id"
  end

  create_table "tasks", force: :cascade do |t|
    t.string "title"
    t.boolean "done"
    t.integer "folder_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.date "deadline"
    t.string "done_by", default: ""
    t.integer "public_folder_id"
    t.index ["folder_id"], name: "index_tasks_on_folder_id"
    t.index ["public_folder_id"], name: "index_tasks_on_public_folder_id"
  end

  create_table "users", force: :cascade do |t|
    t.string "username"
    t.string "password_digest"
```

```
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "email"
  end

  add_foreign_key "folders", "users"
  add_foreign_key "public_folders", "users"
  add_foreign_key "tasks", "folders"
  add_foreign_key "tasks", "public_folders"
end
```

Контроллеры

Текст файла application_controller.rb

```
# frozen_string_literal: true

# ApplicationController
class ApplicationController < ActionController::Base
  include SessionHelper
  before_action :set_locale
  before_action :require_login

  def require_login
    redirect_to session_login_path unless signed_in?
  end

  def default_url_options
    { locale: I18n.locale }
  end

  def set_locale
    I18n.locale = extract_locale || I18n.default_locale
  end

  def extract_locale
    parsed_locale = params[:locale]
    parsed_locale.to_sym if I18n.available_locales.map(&:to_s).include?(parsed_locale)
  end
end
```

Текст файла session_controller.rb

```
# frozen_string_literal: true

# Session Controller
class SessionController < ApplicationController
  skip_before_action :require_login, only: %i[login create]
```

```
  def login; end

  def create
    user = User.find_by_username(params[:username])
    if user&.authenticate(params[:password])
      sign_in user
      redirect_to root_path
    else
      flash[:notice] = if user.nil?
                         t('no_user')
                       else
                         t('wrong_pass')
                       end
      p flash[:notice]
      redirect_to session_login_path
    end
  end

  def logout
    sign_out
    redirect_to session_login_path
  end
end
```

Текст файла task_controller.rb

```
# frozen_string_literal: true

# Controller class for task
class TaskController < ApplicationController
  before_action :raise_error, only: [:output]

  def input; end

  def output
    @max_seg = 0
    str = params[:array]
    arr = str.split(' ').map(&:to_i)
    @segments = get_segments(arr)
    @max_seg = (@segments.max_by { |elem| elem[:length] })[:segment] unless
@segments.nil? || @segments.empty?
    @count = @segments.length
  end

  private

  def perfect?(num)
```

```
    summ = 0
    ans = false
    (1...num).each do |i|
      summ += i if (num % i).zero?
    end
    ans = true if (summ == num) && num.positive?
    ans
  end

  def get_segments(array)
    result = []
    len = 0
    seg = []
    array.each do |elem|
      if perfect?(elem)
        len += 1
        seg.append(elem)
      else
        result.append({ length: len, segment: seg.join(' ') }) if len.positive?
        len = 0
        seg = []
      end
    end
    result.append({ length: len, segment: seg.join(' ') }) if len.positive?
    result
  end

  def raise_error
    if params[:array].nil? || params[:array].empty?
      flash[:error] = 'Ошибка: параметры не должны быть пустыми.'
      redirect_to root_path
    elsif !params[:array].match(/^( ?-?\d)+$/)
      flash[:error] = 'Ошибка: параметры введены некорректно.'
      redirect_to root_path
    end
  end
end
```

Текст файла session_helper.rb

```
module SessionHelper
  def sign_in(user)
    cookies.signed[:user_id] = { value: user.id, expires: 7.days }
    # signed шифрует id пользователя, добавляем срок годности куки, чтобы
пользователя не выкидывало во время сессии
    @current_user = user
  end
```

```
  def signed_in?
    !current_user.nil?
  end

  def sign_out
    cookies.signed[:user_id] = nil
    @current_user = nil
  end

  def current_user
    @current_user ||= User.find_by(id: cookies.signed[:user_id])
    # ||= <=> @current_user = User.find if @current_user.nil?
  end
end
```

Текст файла users_controller.rb

```
# frozen_string_literal: true

# Users Controller
class UsersController < ApplicationController
  before_action :set_user, only: %i[show edit update destroy]
  before_action :check_access, only: %i[show edit update destroy]
  before_action :stay_admin, only: :destroy # cant destroy admin
  skip_before_action :require_login, only: %i[create new]

  # GET /users or /users.json
  def index
    redirect_to root_path unless current_user.id == 6
    @folders = Folder.where(user_id: current_user.id)
    @users = User.all
  end

  # GET /users/1 or /users/1.json
  def show
    @folders = Folder.where(user_id: current_user.id)
  end

  # GET /users/new
  def new
    @user = User.new
  end

  # GET /users/1/edit
  def edit; end
```

```
# POST /users or /users.json
def create
  @user = User.new(user_params)

  respond_to do |format|
    if @user.save
      sign_in @user
      format.html { redirect_to root_path, notice: t('welcome') }
    else
      format.html { render :new, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /users/1 or /users/1.json
def update
  @folders = Folder.where(user_id: current_user.id)
  if @user.update(user_params)
    redirect_to user_url(@user), notice: t('user_was_updated')
  elsif !current_user.nil?
    flash[:error] = 'The task title and deadline cannot be empty :('
    # redirect_to user_url(@user) , status: :unprocessable_entity
    render :show, status: :unprocessable_entity
  else
    redirect_to new_user_path, status: :unprocessable_entity
  end
end

# DELETE /users/1 or /users/1.json
def destroy
  @user.destroy

  respond_to do |format|
    format.html { redirect_to users_url, notice: t('user_was_destroyed') }
  end
end

private

# Use callbacks to share common setup or constraints between actions.
def set_user
  @user = User.find(params[:id])
end

# Only allow a list of trusted parameters through.
def user_params
```

```
    params.require(:user).permit(:username, :password, :password_confirmation, :email)
  end

  def check_access
    redirect_to root_path if (current_user.id != 6) && (current_user != set_user)
  end

  def stay_admin
    redirect_to user_path, alert: t('uradmin') if current_user.id == 6
  end
end
```

Текст файла folders_controller.rb

```
# frozen_string_literal: true

# Controller for Folders
class FoldersController < ApplicationController
  before_action :set_folder, only: %i[show edit update destroy]
  before_action :check_ownership, only: %i[show edit update destroy]

  # GET /folders or /folders.json
  def index
    @folders = Folder.where(user_id: current_user.id)
  end

  # GET /folders/1 or /folders/1.json
  def show
    @folders = Folder.where(user_id: current_user.id)
    @task = Task.new
    @tasks = @folder.tasks
    @colors = ActiveSupport::JSON.decode(@folder.colors)
  end

  # GET /folders/new
  def new
    @folders = Folder.where(user_id: current_user.id)
    @folder = Folder.new
    @colors = ActiveSupport::JSON.decode(@folder.colors)
  end

  # GET /folders/1/edit
  def edit
    @folders = Folder.where(user_id: current_user.id)
    @colors = ActiveSupport::JSON.decode(@folder.colors)
  end
```

```
  # POST /folders or /folders.json
  def create
    red = params[:folder][:red].to_i
    orange = params[:folder][:orange].to_i
    yellow = params[:folder][:yellow].to_i
    colors = { 'red' => red, 'orange' => orange, 'yellow' => yellow }
    colors_json = ActiveSupport::JSON.encode(colors)
    @folder = Folder.new(name: params[:folder][:name], user_id:
params[:folder][:user_id], description: params[:folder][:description], colors:
colors_json)
    if red >= orange || orange >= yellow
      flash[:error] = t('wrong_colors')
      redirect_to new_folder_path
      return
    end

    if @folder.save
      redirect_to folder_url(@folder), notice: t('folder_was_created')
      # format.json { render :show, status: :created, location: @folder }
    else
      flash[:error] = t('folder_name_not_empty')
      redirect_to new_folder_path
      # format.json { render json: @folder.errors, status: :unprocessable_entity }
    end
  end

  # PATCH/PUT /folders/1 or /folders/1.json
  def update
    # colors
    red = params[:folder][:red].to_i
    orange = params[:folder][:orange].to_i
    yellow = params[:folder][:yellow].to_i
    colors = { 'red' => red, 'orange' => orange, 'yellow' => yellow }
    colors_json = ActiveSupport::JSON.encode(colors)
    if red >= orange || orange >= yellow
      flash[:error] = t('wrong_colors')
      redirect_to edit_folder_path
      return
    end

    if @folder.update(name: params[:folder][:name], user_id: params[:folder][:user_id],
description: params[:folder][:description], colors: colors_json) # change to my_params
or add to folder_params colors
      redirect_to folder_url(@folder), notice: t('folder_was_updated')
      # format.json { render :show, status: :ok, location: @folder }
    else
```

```
      flash[:error] = t('folder_name_not_empty')
      redirect_to edit_folder_path
      # format.json { render json: @folder.errors, status: :unprocessable_entity }
    end
  end

  # DELETE /folders/1 or /folders/1.json
  def destroy
    @folder.destroy

    respond_to do |format|
      format.html { redirect_to folders_url, notice: t('folder_was_destroyed') }
      format.json { head :no_content }
    end
  end

  private

  # Use callbacks to share common setup or constraints between actions.
  def set_folder
    @folder = Folder.find(params[:id])
  end

  # Only allow a list of trusted parameters through.
  def folder_params
    params.require(:folder).permit(:name, :user_id, :description) # :colors!!
  end

  def check_ownership
    redirect_to root_path unless @folder.user_id == current_user.id
  end
end
```

Текст файла public_folders_controller.rb

```
# frozen_string_literal: true

# PublicFolders Controller
class PublicFoldersController < ApplicationController
  before_action :set_public_folder, only: %i[show edit update destroy check_ownership]
  before_action :set_owning_folders, only: %i[index show new edit] # ownership for
side bar mapping
  before_action :set_membering_folders, only: %i[index show new edit] # membership
for side bar mapping
  before_action :check_ownership, only: %i[update delete edit] # - only owner can
change info
```

```
  before_action :check_membership, only: %i[show edit] # only member AND owners can do tasks in public folder

  # GET /public_folders or /public_folders.json
  def index; end

  # GET /public_folders/1 or /public_folders/1.json
  def show
    @task = Task.new # empty task for add task form
    @tasks = @public_folder.tasks # all tasks
    @colors = ActiveSupport::JSON.decode(@public_folder.colors) # color settings
    @members = ActiveSupport::JSON.decode(@public_folder.members) # member list
  end

  # GET /public_folders/new
  def new
    @public_folder = PublicFolder.new
    @colors = ActiveSupport::JSON.decode(@public_folder.colors)
    @members = ActiveSupport::JSON.decode(@public_folder.members)
  end

  # GET /public_folders/1/edit
  def edit # hidden field :new_member_id with value: ""
    @colors = ActiveSupport::JSON.decode(@public_folder.colors)
    @members = ActiveSupport::JSON.decode(@public_folder.members)
  end

  # POST /public_folders or /public_folders.json
  def create
    # @public_folder = PublicFolder.new(public_folder_params)

    # respond_to do |format|
    #   if @public_folder.save
    #     format.html { redirect_to public_folder_url(@public_folder), notice: "Public folder was successfully created." }
    #     format.json { render :show, status: :created, location: @public_folder }
    #   else
    #     format.html { render :new, status: :unprocessable_entity }
    #     format.json { render json: @public_folder.errors, status: :unprocessable_entity }
    #   end
    # end

    red = params[:public_folder][:red].to_i
    orange = params[:public_folder][:orange].to_i
    yellow = params[:public_folder][:yellow].to_i
    colors = { 'red' => red, 'orange' => orange, 'yellow' => yellow }
```

```
    colors_json = ActiveSupport::JSON.encode(colors)
    if red >= orange || orange >= yellow
      flash[:error] = t('wrong_colors')
      redirect_to new_public_folder_path
      return
    end

    @members = [] # empty array of member names

    new_member_name = params[:public_folder][:new_member_name]
    if (!User.find_by_username(new_member_name) && (new_member_name != '')) ||
(new_member_name == current_user.username)
      flash[:error] = t 'user_not_found'
      redirect_to new_public_folder_path
      return
    end

    if @members.include?(new_member_name)
      flash[:error] = t 'user_in_members'
      redirect_to new_public_folder_path
      return
    end

    p delete_member_name = params[:public_folder][:delete_member_name]
    if !@members.include?(delete_member_name) && delete_member_name != ''
      flash[:error] = t('user_not_member')
      redirect_to new_public_folder_path
      return
    end

    @members.append(new_member_name) unless new_member_name == '' # adding
new member
    @members.delete(delete_member_name)
    members_json = ActiveSupport::JSON.encode(@members)

    @public_folder = PublicFolder.new(name: params[:public_folder][:name], user_id:
params[:public_folder][:user_id], description: params[:public_folder][:description],
colors: colors_json, members: members_json)
    if red >= orange || orange >= yellow
      flash[:error] = t('wrong_colors')
      redirect_to new_public_folder_path
      return
    end

    if @public_folder.save
      redirect_to public_folder_url(@public_folder), notice: t('folder_was_created')
```

```
    elsif PublicFolder.find_by_name(params[:public_folder][:name])
      flash[:error] = t('folder_name_exists')
      redirect_to new_public_folder_path
    else
      flash[:error] = t('folder_name_not_empty')
      redirect_to new_public_folder_path
    end
  end

  # PATCH/PUT /public_folders/1 or /public_folders/1.json
  def update
    # respond_to do |format|
    #   if @public_folder.update(public_folder_params)
    #     format.html { redirect_to public_folder_url(@public_folder), notice: "Public
folder was successfully updated." }
    #     format.json { render :show, status: :ok, location: @public_folder }
    #   else
    #     format.html { render :edit, status: :unprocessable_entity }
    #     format.json { render json: @public_folder.errors, status: :unprocessable_entity }
    #   end
    # end

    red = params[:public_folder][:red].to_i
    orange = params[:public_folder][:orange].to_i
    yellow = params[:public_folder][:yellow].to_i
    colors = { 'red' => red, 'orange' => orange, 'yellow' => yellow }
    colors_json = ActiveSupport::JSON.encode(colors)
    if red >= orange || orange >= yellow
      flash[:error] = t('wrong_colors')
      redirect_to edit_public_folder_path
      return
    end

    @members = ActiveSupport::JSON.decode(@public_folder.members) # array of
current member names

    new_member_name = params[:public_folder][:new_member_name]
    if (!User.find_by_username(new_member_name) && (new_member_name != '')) ||
(new_member_name == current_user.username)
      flash[:error] = t 'user_not_found'
      redirect_to edit_public_folder_path
      return
    end

    if @members.include?(new_member_name)
      flash[:error] = t 'user_in_members'
```

```
      redirect_to edit_public_folder_path
      return
    end

    p delete_member_name = params[:public_folder][:delete_member_name]
    if !@members.include?(delete_member_name) && delete_member_name != ''
      flash[:error] = t('user_not_member')
      redirect_to edit_public_folder_path
      return
    end

    @members.append(new_member_name) unless new_member_name == '' # adding
new member
    @members.delete(delete_member_name)
    members_json = ActiveSupport::JSON.encode(@members)

    if @public_folder.update(name: params[:public_folder][:name], user_id:
params[:public_folder][:user_id], description: params[:public_folder][:description],
colors: colors_json, members: members_json)
      redirect_to public_folder_url(@public_folder), notice: t('folder_was_updated')
    elsif PublicFolder.find_by_name(params[:public_folder][:name])
      flash[:error] = t('folder_name_exists')
      redirect_to edit_public_folder_path
    else
      flash[:error] = t('folder_name_not_empty')
      redirect_to edit_public_folder_path
    end
  end

  # DELETE /public_folders/1 or /public_folders/1.json
  def destroy
    @public_folder.destroy

    respond_to do |format|
      format.html { redirect_to public_folders_url, notice: t('folder_was_destroyed') }
      format.json { head :no_content }
    end
  end

  private

  # Use callbacks to share common setup or constraints between actions.
  def set_public_folder
    @public_folder = PublicFolder.find(params[:id])
  end
```

```
  def set_owning_folders
    current_user.id
    @owning_folders = PublicFolder.where(user_id: current_user.id) # owning folders
for side bar
  end

  def set_membering_folders # membering folders for side bar, array
    all_public_folders = PublicFolder.all
    p @membering_folders = all_public_folders.select { |folder|
ActiveSupport::JSON.decode(folder.members).include?(current_user.username) }
  end

  def check_ownership
    if current_user.id != @public_folder.user_id
      flash[:notice] = t('must_be_owner')
      redirect_to public_folder_path
    end
  end

  def check_membership
    members = ActiveSupport::JSON.decode(@public_folder.members)
    if !members.include?(current_user.username) && current_user.id !=
@public_folder.user_id
      redirect_to public_folders_path
    end
  end

  # Only allow a list of trusted parameters through.
  def public_folder_params
    params.require(:public_folder).permit(:name, :description, :colors, :members,
:user_id) # colors, members - JSON
  end
end
```

Текст файла tasks_controller.rb

```
# frozen_string_literal: true

# Tasks Controller
class TasksController < ApplicationController
  def index; end

  def new; end

  def show; end

  def create
```

```
    @task = Task.create(task_params) # same as update
    if @task.save
      redirect_to folder_path(@task.folder_id) unless @task.folder_id.nil?
      redirect_to public_folder_path(@task.public_folder_id) unless
@task.public_folder_id.nil?
    else
      flash[:error] = t('task_empty')
      redirect_to folder_path(@task.folder_id) unless @task.folder_id.nil?
      redirect_to public_folder_path(@task.public_folder_id) unless
@task.public_folder_id.nil?
    end
  end

  def update
    @task = Task.find(params[:id])
    if @task.update(task_params)
      redirect_to "/#{locale}/folders/#{@task.folder_id}/#row#{@task.id}" unless
@task.folder_id.nil? # !same as destroy
      unless @task.public_folder_id.nil?
        redirect_to "/#{locale}/public_folders/#{@task.public_folder_id}"
      end # /#row#{@task.id} removed
    else
      redirect_to root_path
    end
  end

  def destroy
    @task = Task.find(params[:id])
    # my_path = folder_path(@task.folder_id) # changed my_path depenging on
public_folder_id existance

    if !@task.folder_id.nil?
      my_path = folder_path(@task.folder_id)
    elsif !@task.public_folder_id.nil?
      my_path = public_folder_path(@task.public_folder_id)
      public_folder = PublicFolder.find(@task.public_folder_id)
      if (public_folder.user_id != current_user.id) && (@task.done_by != '')
        flash[:notice] = t('owner_rule')
        redirect_to "/#{locale}/public_folders/#{@task.public_folder_id}"
        return
      end
    end

    if @task.destroy
      redirect_to my_path
    else
```

```
      redirect_to root_path
    end
  end

  private

  def task_params # migrated to FK folder_id can be null
    params.require(:task).permit(:title, :done, :folder_id, :deadline, :done_by,
:public_folder_id) # added public folder id
  end
end
```

Представления

Текст файлов представления, а также содержание всех остальных файлов можно посмотреть на GitHub: https://github.com/21mad/TaskManager

Тесты моделей

Текст файла folder_spec.rb

```
require 'rails_helper'

RSpec.describe Folder, type: :model do
  subject {
    Folder.new(name: "Cool folder for test", description: "For tests")
  }

  it 'is not valid without user_id' do
    expect(subject).to_not be_valid
  end

  it "is not valid without a name" do
    subject.name = nil
    expect(subject).to_not be_valid
  end

  describe "Associations" do
    it { should belong_to(:user).without_validating_presence }
    it { should have_many(:tasks) }
  end
end
```

Текст файла public_folder_spec.rb

```
require 'rails_helper'

RSpec.describe PublicFolder, type: :model do
```

```
  subject {
    PublicFolder.new(name: "Cool folder for test", description: "For tests")
  }

  it 'is not valid without user_id' do
    expect(subject).to_not be_valid
  end

  it "is not valid without a name" do
    subject.name = nil
    expect(subject).to_not be_valid
  end

  describe "Associations" do
    it { should belong_to(:user).without_validating_presence }
    it { should have_many(:tasks) }
  end
end
```

Текст файла task_spec.rb

```
require 'rails_helper'

RSpec.describe Task, type: :model do
  subject {
    Task.new(title: "Write some tests", deadline: DateTime.now + 3)
  }

  it 'is valid with valid attributes' do
    expect(subject).to be_valid
  end

  it 'is not valid with empty title' do
    subject.title = nil
    expect(subject).not_to be_valid
  end

  it 'is not valid with empty deadline' do
    subject.deadline = nil
    expect(subject).not_to be_valid
  end
end
```

Текст файла user_spec.rb

```
require 'rails_helper'
```

```
RSpec.describe User, type: :model do
  # pending "add some examples to (or delete) #{__FILE__}"
  subject {
    User.new(username: "cool_user", email: "example@mail.ru", password: "password")
  }

  it "is valid with valid attributes" do
    expect(subject).to be_valid
  end

  it 'is not valid with invalid email' do
    subject.email = 'not_an_email@@at.all'
    expect(subject).to_not be_valid
  end

  it "is not valid with short username" do
    subject.username = "user"
    expect(subject).to_not be_valid
  end

  it 'is not valid without an username' do
    subject.username = nil
    expect(subject).to_not be_valid
  end

  it "is not valid without an email" do
    subject.email = nil
    expect(subject).to_not be_valid
  end

  it "is not valid without a password" do
    subject.password = nil
    expect(subject).to_not be_valid
  end

  it 'is not valid with the same username' do
    new_user = User.new(username: "cool_user", email: "newemail123@mail.ru",
password: "password")
    expect(new_user).to be_valid
  end

  it 'is not valid with the same email' do
    new_user = User.new(username: "new_user11", email: "example@mail.ru",
password: "password")
    expect(new_user).to be_valid
  end
```

```
end
```

Системные тесты

Текст файла account_spec.rb

```
require 'rails_helper'
require 'spec_helper'
require 'faker'

RSpec.describe 'User', type: :system do
  before :each do
    create_accs
    User.create(username: 'admin', email: 'admin@example.com', password: 'password')
  end
  context 'Account:' do
    before do
      visit '/en/session/login'
      fill_in :username, with: 'admin'
      fill_in :password, with: 'password'
      click_button 'Log in'
      click_link 'Account'
    end

    scenario 'change your nickname' do
      fill_in :user_username, with: '21mad'
      fill_in :user_password, with: 'admin'
      fill_in :user_password_confirmation, with: 'admin'
      click_button 'Update'
      expect(page).to have_content("User was successfully updated.")
      expect(page).to have_content("21mad")
    end

    scenario 'change your password' do
      fill_in :user_username, with: 'admin'
      fill_in :user_password, with: 'new_password'
      fill_in :user_password_confirmation, with: 'new_password'
      click_button 'Update'
      expect(page).to have_content("User was successfully updated.")
      expect(page).to have_content("admin")
    end

                scenario 'cannot delete account if admin' do
      click_button 'Delete this account'
                        expect(page).to have_content("You're admin!")
    end
```

```
    scenario 'can delete account if user' do
      User.create(username: 'just_user', email: 'just_user@mail.ru' , password: 'password')
      click_link 'Logout'
      fill_in :username, with: 'just_user'
      fill_in :password, with: 'password'
      click_button 'Log in'
      click_link 'Account'
      click_button 'Delete this account'
                          expect(page).to have_content("Account was successfully deleted.")
    end

  end
end



def create_accs
  5.times{ User.create(username: Faker::Lorem.name, email: (Faker::Lorem.word +
'@mail.ru') , password: 'password') }
  User.create(username: 'just_user1', email: 'just_user1@mail.ru' , password: 'password')
  User.create(username: 'just_user2', email: 'just_user2@mail.ru' , password: 'password')
  User.create(username: 'just_user3', email: 'just_user3@mail.ru' , password: 'password')
end
```

Текст файла all_users_spec.rb

```
require 'rails_helper'
require 'spec_helper'
require 'faker'

RSpec.describe 'Users', type: :system do
  before :each do
    create_accs
    User.create(username: 'admin', email: 'admin@example.com', password: 'password')
  end
  context 'All Users:' do
    before do
      visit '/en/session/login'
      fill_in :username, with: 'admin'
      fill_in :password, with: 'password'
      click_button 'Log in'
    end

    scenario 'admin can see user list' do
      click_link 'All Users'
      expect(page).to have_content("User list")
      expect(page).to have_content("User name: admin")
    end
```

```
  scenario 'admin can see users accounts' do
    click_link 'All Users'
    first(".user_link").click
    expect(page).to have_content('User Information')
  end

  scenario 'user cannot see user list' do
    User.create(username: 'just_user', email: 'just_user@mail.ru' , password: 'password')
    click_link 'Logout'
    fill_in :username, with: 'just_user'
    fill_in :password, with: 'password'
    click_button 'Log in'
          expect(page).to_not have_content("All Users")
  end

 end
end



def create_accs
 5.times{ User.create(username: Faker::Lorem.name, email: (Faker::Lorem.word +
'@mail.ru') , password: 'password') }
 User.create(username: 'just_user1', email: 'just_user1@mail.ru' , password: 'password')
 User.create(username: 'just_user2', email: 'just_user2@mail.ru' , password: 'password')
 User.create(username: 'just_user3', email: 'just_user3@mail.ru' , password: 'password')
end
```

Текст файла my_tasks_spec.rb

```
require 'rails_helper'
require 'spec_helper'

RSpec.describe 'Folders', type: :system do
 before :each do
  User.create(username: 'admin', email: 'admin@example.com', password: 'password')
 end
 context 'My Tasks:' do
  before do
    visit '/en/session/login'
    fill_in :username, with: 'admin'
    fill_in :password, with: 'password'
    click_button 'Log in'
    click_link 'My Tasks'
  end

  feature 'Creating new folder' do
```

```
  before do
    click_button 'Create'
    fill_in :folder_name, with: 'valid name'
    fill_in :folder_description, with: 'description'
    fill_in :folder_red, with: 1
    fill_in :folder_orange, with: 3
    fill_in :folder_yellow, with: 6
  end

  scenario 'create new folder with valid params' do
    click_button 'Create a Folder'
    expect(page).to have_content("Folder was successfully created.")
    expect(page).to have_content("valid name")
  end

  scenario 'create new folder with empty name' do
    fill_in :folder_name, with: ''
    click_button 'Create a Folder'
    expect(page).to have_content("Folder name cannot be empty.")
  end

  scenario 'create new folder with wrong colors' do
    fill_in :folder_red, with: 4
    fill_in :folder_orange, with: 2
    fill_in :folder_yellow, with: 6
    click_button 'Create a Folder'
    expect(page).to have_content("Wrong color settings.")
  end
end

feature 'Deleting folder' do
  before do
    click_button 'Create'
    fill_in :folder_name, with: 'valid name'
    fill_in :folder_description, with: 'description'
    fill_in :folder_red, with: 1
    fill_in :folder_orange, with: 3
    fill_in :folder_yellow, with: 6
    click_button 'Create a Folder'
  end

  scenario 'delete a folder' do
    first(".bi > img").click
    click_button 'Delete this folder!'
    expect(page).to have_content("Folder was successfully destroyed.")
  end
```

```
  end

  feature 'Tasks' do
    before do
      click_button 'Create'
      fill_in :folder_name, with: 'valid name'
      fill_in :folder_description, with: 'description'
      fill_in :folder_red, with: 1
      fill_in :folder_orange, with: 3
      fill_in :folder_yellow, with: 6
      click_button 'Create a Folder'
      first('.nav-link > a:nth-child(2)').click
    end

    scenario 'adding new task with valid params' do
      fill_in :task_title, with: 'test_task'
      fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
      click_button 'Add task'
      expect(page).to have_content("test_task")
    end

    scenario 'adding new task with empty title' do
      fill_in :task_title, with: ''
      fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
      click_button 'Add task'
      expect(page).to have_content("The task title and deadline cannot be empty :(")
    end

    scenario 'adding new task with empty deadline' do
      fill_in :task_title, with: 'test_task'
      click_button 'Add task'
      expect(page).to have_content("The task title and deadline cannot be empty :(")
    end

    scenario 'delete task' do
      fill_in :task_title, with: 'test_task'
      fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
      click_button 'Add task'
      find('#row1 .button_to img').click
      expect(page).to have_content("This folder is empty :(")
    end
  end
 end
end
```

Текст файла public_tasks_spec.rb

```
require 'rails_helper'
require 'spec_helper'

RSpec.describe 'Public Folders', type: :system do
  before :each do
    User.create(username: 'admin', email: 'admin@example.com', password: 'password')
  end
  context 'My Tasks:' do
    before do
      visit '/en/session/login'
      fill_in :username, with: 'admin'
      fill_in :password, with: 'password'
      click_button 'Log in'
      click_link 'Public Tasks'
    end

    feature 'Creating new public folder: ' do
      before do
        click_button 'Create'
        fill_in :public_folder_name, with: 'valid name'
        fill_in :public_folder_description, with: 'description'
        fill_in :public_folder_red, with: 1
        fill_in :public_folder_orange, with: 3
        fill_in :public_folder_yellow, with: 6
      end

      scenario 'create new public_folder with valid params' do
        click_button 'Create a Public folder'
        expect(page).to have_content("Folder was successfully created.")
        expect(page).to have_content("valid name")
      end

      scenario 'create new public_folder with empty name' do
        fill_in :public_folder_name, with: ''
        click_button 'Create a Public folder'
        expect(page).to have_content("Folder name cannot be empty.")
      end

      scenario 'create new public_folder with wrong colors' do
        fill_in :public_folder_red, with: 4
        fill_in :public_folder_orange, with: 2
        fill_in :public_folder_yellow, with: 6
        click_button 'Create a Public folder'
        expect(page).to have_content("Wrong color settings.")
      end
```

```
  end

  feature 'Deleting public folder: ' do
    before do
      click_button 'Create'
      fill_in :public_folder_name, with: 'valid name'
      fill_in :public_folder_description, with: 'description'
      fill_in :public_folder_red, with: 1
      fill_in :public_folder_orange, with: 3
      fill_in :public_folder_yellow, with: 6
      click_button 'Create a Public folder'
    end

    scenario 'delete a public folder' do
      first(".bi > img").click
      click_button 'Delete this folder!'
      expect(page).to have_content("Folder was successfully destroyed.")
    end
  end

  feature 'Adding and deleting members: ' do
    before do
      click_button 'Create'
      fill_in :public_folder_name, with: 'valid name'
      fill_in :public_folder_description, with: 'description'
      fill_in :public_folder_red, with: 1
      fill_in :public_folder_orange, with: 3
      fill_in :public_folder_yellow, with: 6
      click_button 'Create a Public folder'
      first(".bi > img").click
      User.create(username: 'new_member', email: 'member@example.com', password:
'password')
    end

    scenario 'adding a new member with correct nickname' do
      fill_in :public_folder_new_member_name, with: 'new_member'
      click_button 'Update a Public folder'
      expect(page).to have_content('Folder was successfully updated.')
    end

    scenario 'adding a person which is member already' do
      fill_in :public_folder_new_member_name, with: 'new_member'
      click_button 'Update a Public folder'
      first(".bi > img").click
      fill_in :public_folder_new_member_name, with: 'new_member'
      click_button 'Update a Public folder'
```

```
    expect(page).to have_content('User to add is already in member list.')
  end

  scenario 'adding a person with incorrect nickname' do
    fill_in :public_folder_new_member_name, with: 'new_memb eeer'
    click_button 'Update a Public folder'
    expect(page).to have_content('User to add not found.')
  end

  scenario 'deleting a member with correct nickname' do
    fill_in :public_folder_new_member_name, with: 'new_member'
    click_button 'Update a Public folder'
    first(".bi > img").click
    fill_in :public_folder_delete_member_name, with: 'new_member'
    click_button 'Update a Public folder'
    expect(page).to have_content('Folder was successfully updated.')
  end

  scenario 'deleting a member with incorrect nickname' do
    fill_in :public_folder_new_member_name, with: 'new_member'
    click_button 'Update a Public folder'
    first(".bi > img").click
    fill_in :public_folder_delete_member_name, with: 'new_memb eeer'
    click_button 'Update a Public folder'
    expect(page).to have_content('User to delete is not in the list of members.')
  end

end

feature 'Tasks: ' do
  before do
    click_button 'Create'
    fill_in :public_folder_name, with: 'valid name'
    fill_in :public_folder_description, with: 'description'
    fill_in :public_folder_red, with: 1
    fill_in :public_folder_orange, with: 3
    fill_in :public_folder_yellow, with: 6
    click_button 'Create a Public folder'
    first('.nav-link > a:nth-child(2)').click
  end

  scenario 'adding new task with valid params' do
    fill_in :task_title, with: 'test_task'
    fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
    click_button 'Add task'
    expect(page).to have_content("test_task")
```

```
    end

    scenario 'adding new task with empty title' do
      fill_in :task_title, with: ''
      fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
      click_button 'Add task'
      expect(page).to have_content("The task title and deadline cannot be empty :(")
    end

    scenario 'adding new task with empty deadline' do
      fill_in :task_title, with: 'test_task'
      click_button 'Add task'
      expect(page).to have_content("The task title and deadline cannot be empty :(")
    end

    scenario 'delete task' do
      fill_in :task_title, with: 'test_task'
      fill_in :task_deadline, with: DateTime.current.strftime("%m%d%Y\t%I%M%P")
      click_button 'Add task'
      find('#row1 .button_to img').click
      expect(page).to have_content("This folder is empty :(")
    end
  end
 end
end
```

Текст файла users_login_spec.rb

```
require 'rails_helper'
RSpec.describe 'Session', type: :system do
 feature 'User logs in and logs out' do
   scenario 'with correct details', js: true do
     User.create(username: 'test_user', email: 'user@example.com', password:
'password')

    visit '/en/session/login'

    fill_in :username, with: 'test_user'
    fill_in :password, with: 'password'
    click_button 'Log in'
    expect(current_path).to eq '/en'
    expect(page).to have_content 'Choose an existing folder or create a new one!'

    click_link 'Logout'
    expect(current_path).to eq '/en/session/login'
    expect(page).to have_content 'To continue, log in to your account:'
  end
```

```
    scenario 'Check without register' do
      routes = ['/en', '/en/public_folders', '/en/folders']
      routes.each do |path|
        to_path(path)
      end
    end

    def to_path(path)
      visit path
      expect(current_path).to eq '/en/session/login'
    end
  end
end
```

Результаты выполнения всех тестов

```
""
[]
[]
[]
[]
........

Finished in 25.84 seconds (files took 1.18 seconds to load)
83 examples, 0 failures
```

Вывод

В рамках прохождения курса «Языки Интернет-программирования» получил базовые знания о языках HTML, CSS, JavaScript, изучил основные концепции разработки и тестирования веб-приложений при помощи средств Ruby on Rails. Выполнив данную работу, на практике закрепил полученные знания и навыки, научился правильно искать нужную информацию по разработке, изучил множество документаций различных механизмов, доступных в Rails. Проделанная работа действительно открыла для меня много нового и интересного.