Титулник

Содержание

1 Постановка задачи

Цель: разработка веб-сервиса для загрузки, обработки и сохранения изображений.

Приложение считается MVP только при реализации функционала загрузки изображения, простейшей цветокоррекции, изменения размеров первоначального изображения и скачивания обработанного изображения на устройство.

Сервис будет состоять из backend микросервисов, frontend SPA, веб-сервера и базы данных.

Данный веб-сервис позволит быстро отредактировать изображения без необходимости скачивать специализированные приложения на устройство.

2 Методы решения задачи

На стадии планирования архитектуры будущего приложения, исходя из опыта использования приложений для редактирования изображений по типу GIMP и Adobe Photoshop, было выдвинуто несколько требований к сервису:

— Корректирование яркости, контрастности, насыщенности, сепии, градиентов серого и инверсии.

— Изменение размеров выбором области, которую необходимо сохранить.

— Сжатие изображения по алгоритму JPEG с возможностью выбора качества сжатия и просмотра текущего и будущего размеров файла.

— Наложение различных эффектов поверх изображения.

— Использование системы слоёв, чтобы можно было вернуть изображение до применения изменений.

— Регистрация, авторизация и аутентификация пользователей с возможностью их разделения на отдельные группы (студенты, дизайнеры).

3 Стек технологий

Для выполнения поставленных требований были выбраны следующие средства разработки:

3.1 Серверная сторона

3.1.1 Сервис для регистрации, аутентификации и авторизации

— Язык программирования Python 3.11.

— Фреймворк FastAPI.

— Библиотека psycopg2 для взаимодействия с базой данных.

— Библиотека pydantic для описания моделей базы данных.

— Библиотека PyJWT для работы с jwt токенами.

— ASGI-сервер uvicorn.

3.1.2 Сервис для редактирования изображений

— Язык программирования Python 3.11.

— Фреймворк Flask.

— Библиотеки Pillow и OpenCV-python для обработки изображений.

3.2 Клиентская сторона

— Инструмент сборки Vite.

— Фреймворк Vue.js 3.

— Язык гипертекстовой разметки HTML5.

— Язык каскадных таблиц стилей CSS3.

— Язык программирования JavaScript ES6.

3.3 База данных

— Система управления базами данных PostgreSQL.

3.4 Запуск

— Веб-сервер Nginx.

— Контейнеризатор приложений Docker.

— Средство создания микрослужб Docker Compose.

4 Описание структуры учебного проекта

В проекте находятся папки “backend”, “db”, “frontend”, “nginx”, а также файлы “.gitignore”, “docker-compose.yml” и “README.md”. Рассматривать структуры будем в порядке их нахождения в корневой папке проекта.

4.1 Краткое описание общей структуры

4.1.1 “backend”

Состоит из двух микросервисов:

- “auth” – микросервис для регистрации, авторизации и аутентификации
- “editor” – микросервис для редактирования изображений

4.1.2 “db”

Содержит 3 части:

- “data” – том для сохранения состояния бд (будет подробнее прокомментирован при рассмотрении “docker-compose.yml”).
- “init” – папка с SQL-скриптами для инициализации бд.
- “database.env” – файл с переменными окружения бд.

4.1.3 “frontend”.

Структура автоматически сгенерирована при помощи Vite, рассмотрим только основные части:

- “src” – содержит компоненты, миксины, плагины, статические ресурсы, vuex store, роутинг, главный компонент “App.vue”, главный файл инициализации проекта “main.js” и файл с глобальными стилями “style.css”.
- “Dockerfile” – содержит описание контейнера фронтенда.

— “favicon.ico” – собственноручно спроектированное лого проекта.
— “index.html” – главный html-файл.
— “package.json” – описание всех зависимостей frontend приложения.
— “vite-config.js” – конфигурация Vite.

4.1.4 “nginx”

Содержит 2 файла:

— “Dockerfile” – описание контейнера nginx.
— “nginx.conf” – конфигурационный файл, описывающий работу веб-сервера.

4.1.5 Прочее

“.gitignore” – содержит описание файлов, которых системе контроля версий не стоит сохранять.

“docker-compose.yml” – инструкции для развёртывания служб.

“README.md” – описание проекта для github. Ссылка на проект.

Теперь рассмотрим некоторые элементы подробнее.

4.2 Описание работы серверной части

4.2.1 Сервис для регистрации, авторизации и аутентификации

4.2.1.1 API

Данный микросервис имеет префикс “/api/auth” и предоставляет следующее API:

— “/user/signup” – принимает на вход класс User, который представляет из себя описание модели всех данных пользователя:

```
class User(BaseModel):
    login: str = Field(default=None)
    password: str = Field(default=None)
    name: str = Field(default=None)
    surname: str = Field(default=None)
    patronymic: str = Field(default='')
    email: EmailStr = Field(default=None)
    birthdate: str = Field(default=None)
    category: int = Field(default=None)
```

Далее хеширует пароль пользователя и вставляет его в базу данных, возвращая при этом сгенерированный по почте пользователя jwt токен.

— “/user/check” – принимает на вход класс UserLogin, который содержит только логин или почту пользователя, а также его пароль. Проверяет существует ли такой пользователь в базе данных. Возвращает результат проверки поступившего на вход функции пароля и пароля из базы данных.

— “/user/login” – принимает вход класс UserLogin, проверяет его предыдущей функцией и в случае успешной проверки возвращает jwt токен сгенерированный на основе логина или почты, иначе выкидывает 403 ошибку.

— “/user/profile” – декодирует входной jwt токен и по нему возвращает пользователя из базы данных.

— “/user/logout” – возвращает пустой набор данных для обновления состояния пользователя на клиентской стороне.

4.2.1.2 Работа с БД

Основная работа с БД реализована в “PostgrecConnection.py”, описание моделей в “data_request_model.py”.

— Сначала мы создаём экземпляр класса “PostgreConn” и загружаем в него настройки для работы с БД. Настройки представляют из себя host, имя и пароль пользователя бд, а также название самой бд.

— Основная функция “add”, который создаёт само подключение к бд и выполняет SQL скрипт, который мы в него передаём. Если скрипт начинается со слова “SELECT”, то мы возвращаем его результат выполнения, иначе возвращаем слово “done”.

— Функция “select_user” принимает на вход “userID” представляющий из себя либо логин, либо почту и возвращающий полный набор данных модели “User” за исключением пароля, а также номера категории, потому что мы сразу, используя “JOIN” возвращаем само название категории из таблицы с категориями.

— Функция “insert_user” принимает на вход модель “User”. Сначала проверяет есть ли уже такой пользователь в бд и если есть, то возвращает информацию о том, что такой пользователь уже зарегистрирован, иначе добавляет пользователя в бд и возвращает “done”.

— Функция “check” принимает на вход модель UserLogin и выполняет SELECT-запрос по данным пользователя.

4.2.1.3 Работа с jwt токеном и хеширование

Логика находится в пакете “jwt_utils”.

— “.env” - хранит в себе информацию о секретном ключе и алгоритме jwt.

“jwt_handler.py”:

— “signJWT” - кодирует “userID” и время жизни в jwt токен.

— “decodeJWT” - декодирует из jwt токена информацию о “userID” и время жизни токена. Если время истекло, то возвращает None.

“jwt_bearer” – представляет из себя класс наследник HTTPBearer и реализует следующую логику:

— “__call__” - принимает на вход запрос и возвращает объект “credentials”, представляющий из себя данные авторизации, содержащиеся в заголовке запроса по схеме “Bearer”.
— “verify_jwt” - декодирует токен и проверяет его валидность.

“pass_hash” – реализует логику безопасного хранения пароля, используя экземпляр класса CryptContext по схеме bcrypt:

— “hash” – возвращает хешированный пароль
— “verify” сравнивает переданный не хешированный пароль и пароль из бд, который является хешированным.

4.2.2 Сервис для обработки изображений

В самом начале определяются CORS заголовки для обеспечения правильной обработки запросов. А также “defaultdict” из библиотеки “collections” для хранения редакторов по ключам (логинам пользователей). Сам редактор представляет из себя класс, хранящий в себе изображение.

4.2.2.1 API

Данный микросервис имеет префикс “/api/editor” и предоставляет следующее API:

— “/upload” – создаёт редактор по переданному ему изображению и добавляет его в словарь редакторов, сразу парсит все переданные методы (будет рассмотрено в следующем пункте).

— “/compress_size” – берёт редактор по логину из словаря редакторов и возвращает сжатую картинку и её размер.

— “/pre_prikol” – применяет эффект и возвращает картинку с эффектом, а также путь к файлу с эффектом.

4.2.2.2 Работа редактора изображений

Процесс работы редактора:

— Создаётся экземпляр класса “Editor” из “editor_module.py”. В нём хранится картинка преобразованная из base64 в PIL image.

— Используется один из методов (parse/get_prikol/get_compress).

— “parse” – проходит по модулям “color”, “size”, “prikols” и также применяет к ним “parse”. В этом методе в каждом из модулей проверятся отличается ли значение переданных методов обработки изображения и если отличается, то эффекты применяются.

— “get_prikol” – обращается к модулю “prikols” и возвращает картинку и путь к файлу с эффектом.

— “base64_to_pil” и “pil_to_base64” преобразуют из base64 в формат PIL и обратно соответственно. Листинг кода:

```
    @staticmethod
    def base64_to_pil(img_base64):
        """
        Convert base64 image data to PIL image
        """
        image_data    =    re.sub('^',    '',
img_base64)
        pil_image                                                 =
Image.open(BytesIO(base64.b64decode(image_data)))
        if pil_image.mode != "RGB":
            pil_image = pil_image.convert("RGB")
        return pil_image

    @staticmethod
    def pil_to_base64(img, rate=100):
        buffered = BytesIO()
```

```
        img.save(buffered, format="JPEG", quality=rate)
        buffered.seek(0)
        img_byte = buffered.getvalue()
        return             ""             +
base64.b64encode(img_byte).decode()
```

Рассмотрим работу каждого модуля:

— “color.py”. В яркости, насыщенности, контрастности и градиентах серого используется ImageEnchance. Он накладывает соответствующий рассматриваемому методу эффект с переданным фактором. Для сепии и инверсии используется ImageOps и Image.blend для смешивания переданного изображения и изображения с наложенным эффектом. Для фильтра сепии используется функция “sepia_filter(img)”, в которой специально подобраны коэффициенты, чтобы совпадать с “filter: sepia()” в CSS3. Листинг кода:

```
def parse(img, methods):
    for method, value in methods.items():
        match method:
            case 'brightness':
                if value != 100:
                    brightness_factor = value / 100
                    # Фактор яркости (больше 1 увеличит яркость,
меньше 1 уменьшит)
                    enhancer = ImageEnhance.Brightness(img)
                    img = enhancer.enhance(brightness_factor)
            case 'saturation':
                if value != 100:
                    saturation_factor = value / 100
                    #  Фактор  насыщенности  (больше  1  увеличит
насыщенность, меньше 1 уменьшит)
                    enhancer = ImageEnhance.Color(img)
                    img = enhancer.enhance(saturation_factor)
            case 'contrast':
                if value != 100:
                    contrast_factor = value / 100
                    #  Фактор  контрастности  (больше  1  увеличит
контрастность, меньше 1 уменьшит)
                    enhancer = ImageEnhance.Contrast(img)
                    img = enhancer.enhance(contrast_factor)
            case 'sepia':
                if value != 0:
                    sepia_image = sepia_filter(img)
                    img = Image.blend(img, sepia_image, value/100)
            case 'grayscale':
```

```
                if value != 0:
                    enhancer = ImageEnhance.Color(img)
                    # Преобразование в оттенки серого с настройкой
уровня (0.0 - полностью чёрно-белое, 1.0 - без изменений)
                    img = enhancer.enhance(1 - value / 100)
            case 'invert':
                if value != 0:
                    inverted_image = ImageOps.invert(img)
                    # Инвертирование с настройкой уровня (0.0 -
без изменений, 1.0 - полностью инвертировано)
                    img = Image.blend(img, inverted_image, value /
100)
    return img


def sepia_filter(img):
    width, height = img.size
    sepia_data = []
    for y in range(height):
        for x in range(width):
            r, g, b = img.getpixel((x, y))
            tr = int(0.393 * r + 0.769 * g + 0.189 * b)
            tg = int(0.349 * r + 0.686 * g + 0.168 * b)
            tb = int(0.272 * r + 0.534 * g + 0.131 * b)
            sepia_data.append((tr, tg, tb))

    # Create a new image with the sepia data
    sepia_image = Image.new('RGB', (width, height))
    sepia_image.putdata(sepia_data)

    # Adjust the intensity
    return sepia_image.point(lambda i: i)
```

— “size.py”. Извлекаем x, y самой левой верхней точки и ширину, высоту требуемых размеров изображения. Листинг кода:

```
def parse(img, methods):
 (x, w, y, h) = methods
 crop = (x, y, w, h)
 if (x == 0 and y == 0 and w == img.size[0] and h == img.size[1]\
         or crop == (0, 0, 0, 0)):
     return img
 img_crop = Image.new(color=0, size=(w - x, h - y), mode="RGB")
 img_crop.paste(img.crop(crop), (0, 0, w - x, h - y))
 img_crop.save('cur_size.jpg', 'JPEG')
 return img_crop
```

— “prikols”. Определяем метод эффектов и накладываем его. Для “D̷R̷A̷I̷N̷ S̷T̷Y̷L̷E̷” используется случайная картинка из папки “./drain_gang_effects/”. Для “𝕖𝕝𝕖𝕘𝕒𝕟𝕥” используется текстура зернистости и виньетка. Для “sudo rm -fr /background --no-preserve-root” используется модуль OpenCV, который вырезает фон изображения. Для наложения одного изображения на другое используется функция “overlay_images()”, которая накладывает их друг на друга с помощью “ImageChops.screen()”. Листинг кода:

```
import base64
import os
import cv2
import numpy as np
from PIL import Image, ImageChops, ImageDraw, ImageFilter, ImageOps
import random


def parse(img, method):
    prikol = method['prikol']
    file = method['file']
    return base(img, prikol, file)[0]


def base(img, prikol, file):
    match prikol:
        case                              'D/
S-
            img                 =                overlay_images(img,
f'drain_gang_effects/{file}')
        case '𝕖𝕝𝕖𝕘𝕒𝕟𝕥':
            img = overlay_images(img, 'elegant/zern.jpg')
            img = apply_vignette(img)
        case 'sudo rm -fr /background --no-preserve-root':
            img = remove_background()
    return img, file


def pre_parse(img, method):
    file = random.choice(os.listdir('drain_gang_effects'))
    method = method['prikol']
    return base(img, method, file)


def overlay_images(img, overlay_path):
    background = img.convert('RGB')
    overlay = Image.open(overlay_path).convert('RGB')

    if background.size != overlay.size:
```

```
        overlay = overlay.resize(background.size)

    blended_image = ImageChops.screen(background, overlay)

    blended_image.save('cur_prikols.jpg')
    return blended_image


def remove_background():
    image = cv2.imread('cur.jpg')
    # Create a mask initialized with zeros
    mask = np.zeros(image.shape[:2], np.uint8)

    # Define the rectangle enclosing the object of interest
    rectangle = (10, 10, image.shape[1] - 30, image.shape[0] - 30)

    # Apply GrabCut algorithm to segment the image and refine the
mask
    cv2.grabCut(image,   mask,   rectangle,   None,   None,   5,
cv2.GC_INIT_WITH_RECT)

    # Create a mask where foreground and possible foreground pixels
are marked as likely
    mask_likely  =  np.where((mask  ==  2)  |  (mask  ==  0),  0,
1).astype('uint8')

    # Apply the mask to the original image
    result_image = cv2.bitwise_and(image, image, mask=mask_likely)

    rgb_image = cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB)
    return Image.fromarray(rgb_image)


def apply_vignette(image):
    # Create a circular mask with a radial gradient
    mask = Image.new('L', image.size, 0)
    draw = ImageDraw.Draw(mask)
    draw.ellipse((0,  0,  image.width,  image.height),  fill=255,
outline=0)
    vignette = ImageOps.fit(mask, (image.width, image.height))

    # Apply blur effect to the vignette mask
    blurred_vignette                                          =
vignette.filter(ImageFilter.GaussianBlur(radius=image.width/10))

    # Apply the blurred vignette mask to the image
    result = Image.new('RGB', image.size)
    result.paste(image, mask=blurred_vignette)

    return result
```

4.3 Описание работы клиентской части

Клиентская часть представляет из себя SPA реализованное с помощью фреймворка Vue.js.

4.3.1 Внешний вид и роутинг

— “/”. Главная страница пока пользователь не зарегистрирован выглядит следующим образом:

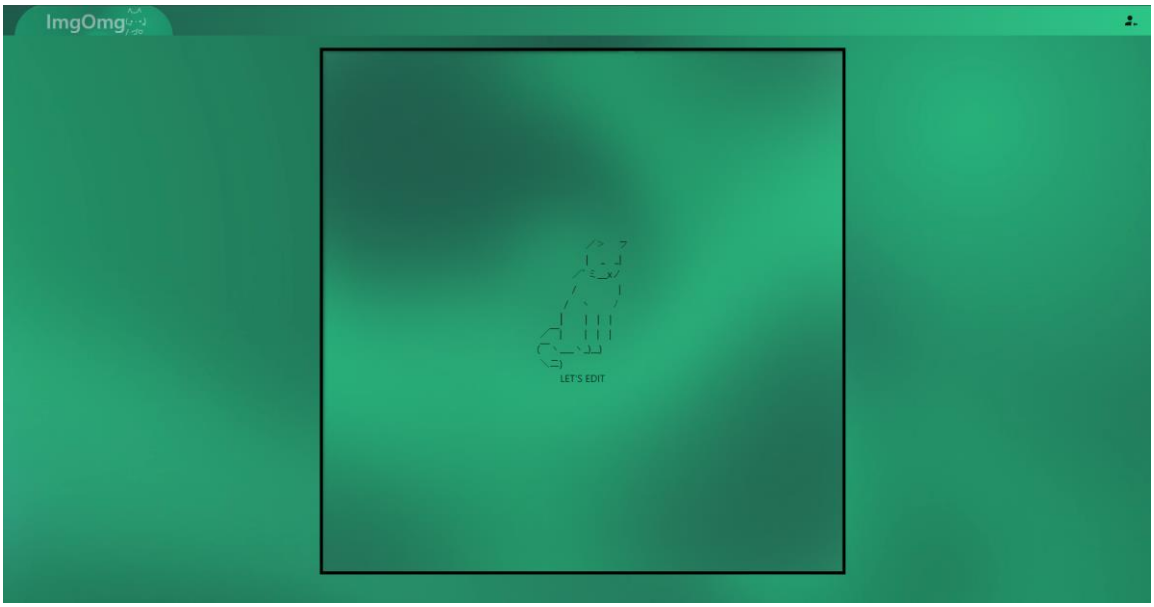


Рисунок 1. Главный экран неавторизированного пользователя

Если нажать на большую квадратную кнопку по центру экрана или на иконку справа навигационной панели, то появится окно регистрации/авторизации:

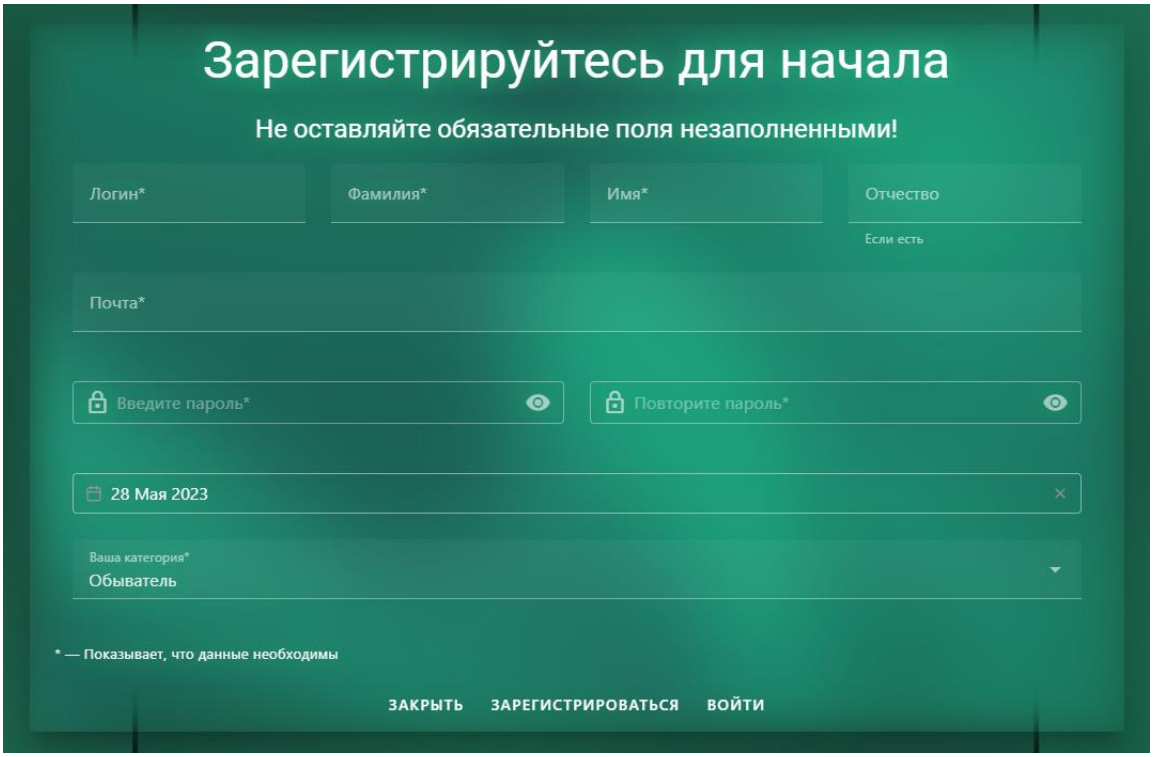


Рисунок 2. Меню регистрации

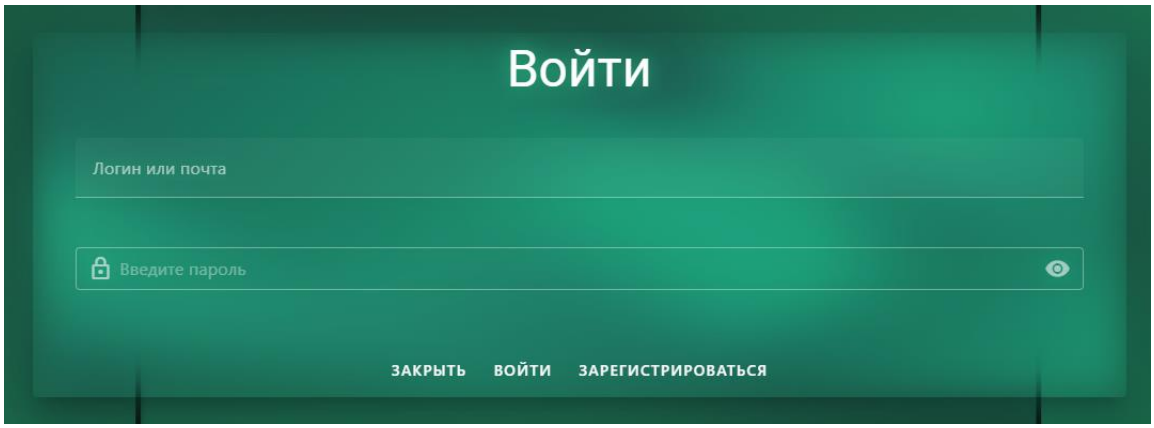
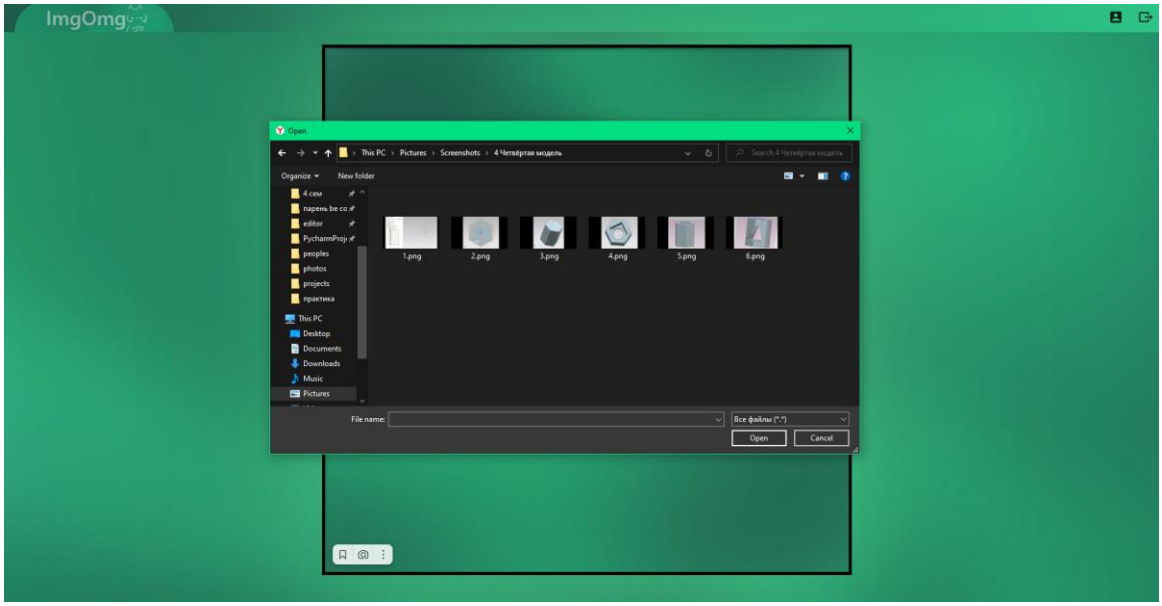


Рисунок 3. Меню входа

После входа/регистрации появится 2 новых значка на навигационной панели и можно повторно нажать на большую квадратную кнопку по центру экрана, что приведёт к открытию формы выбора изображения.



Как только пользователь выберет изображение для редактирования откроется основное меню, где и предстоит обрабатывать изображение.

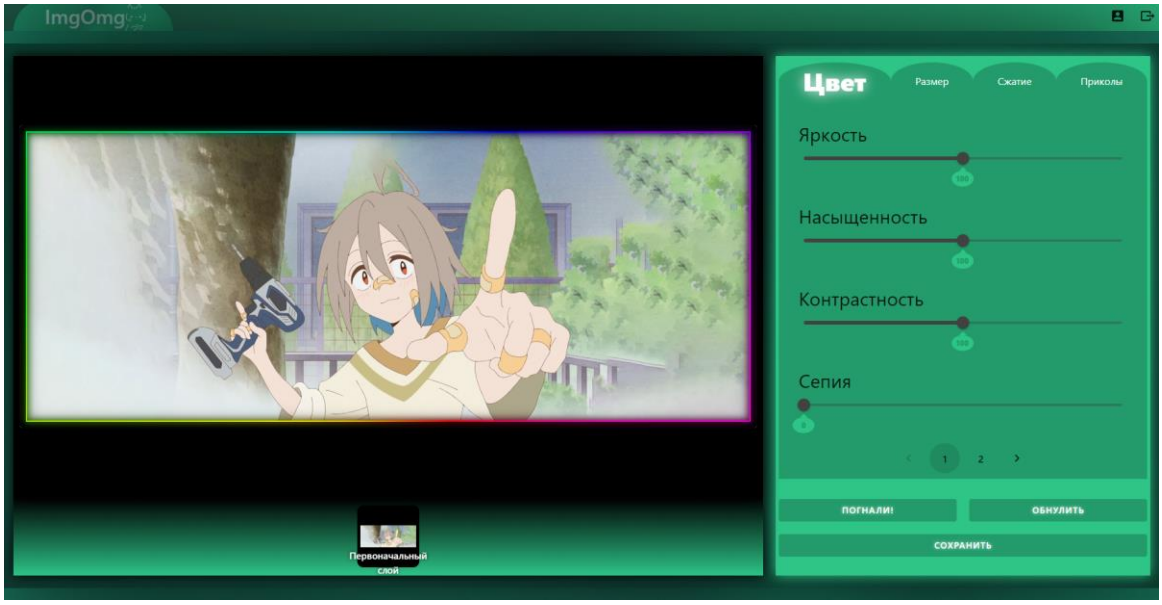


Рисунок 4. Редактор для широкого экрана

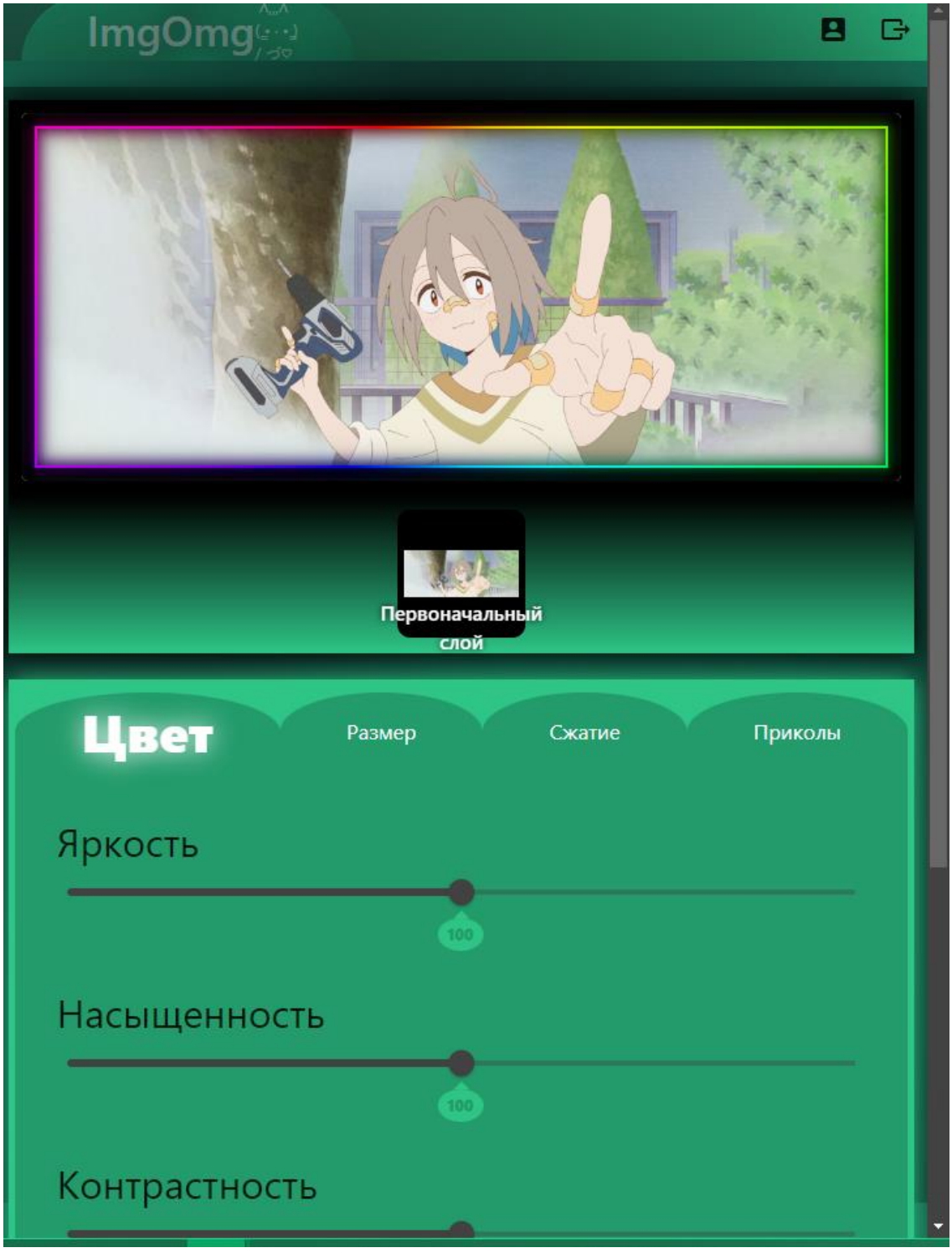


Рисунок 5. Редактор для мобильных устройств

Все панели для редактирования:

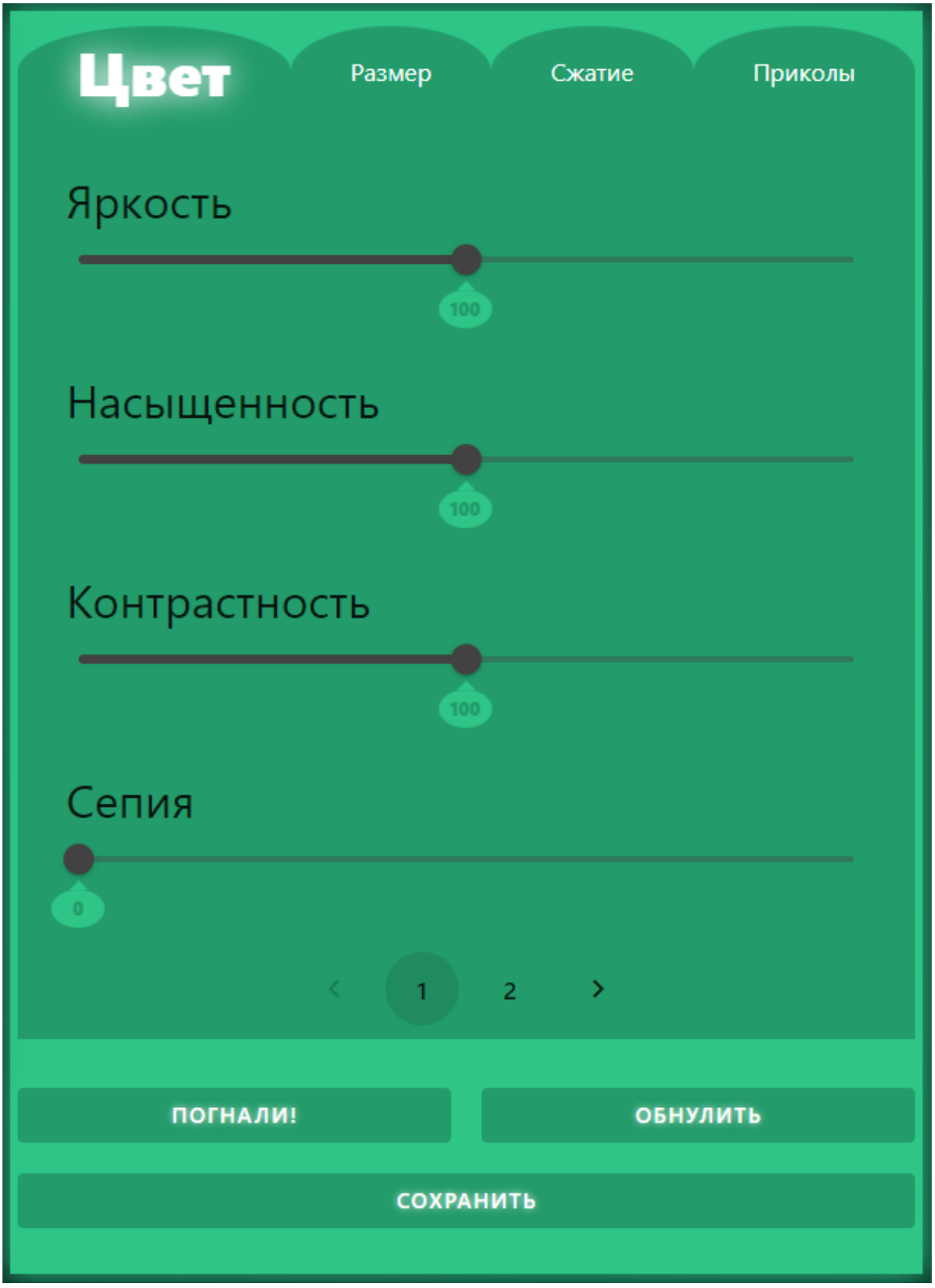


Рисунок 6. Панель редактирования цвета

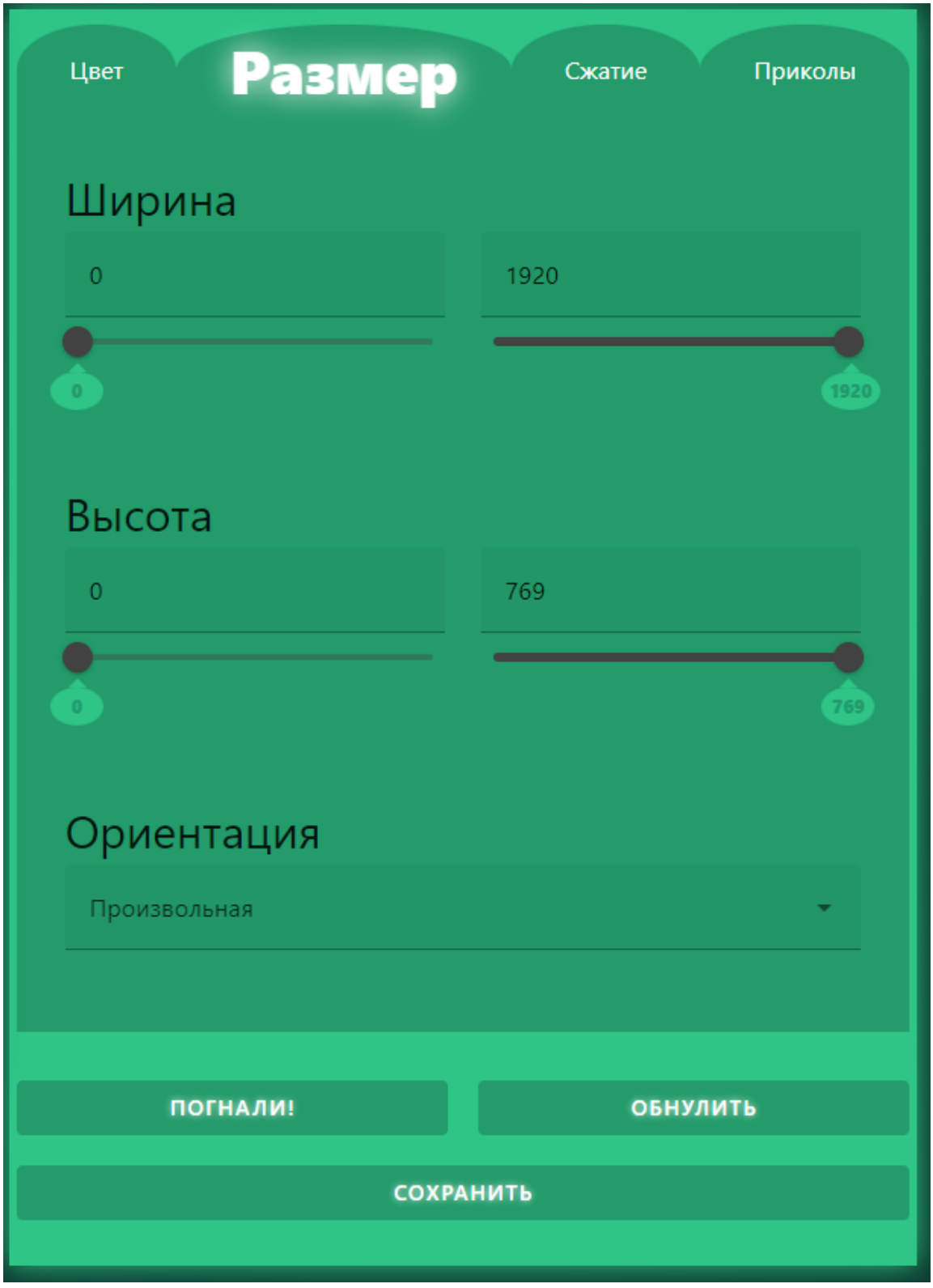


Рисунок 7. Панель редактирования размеров изображения

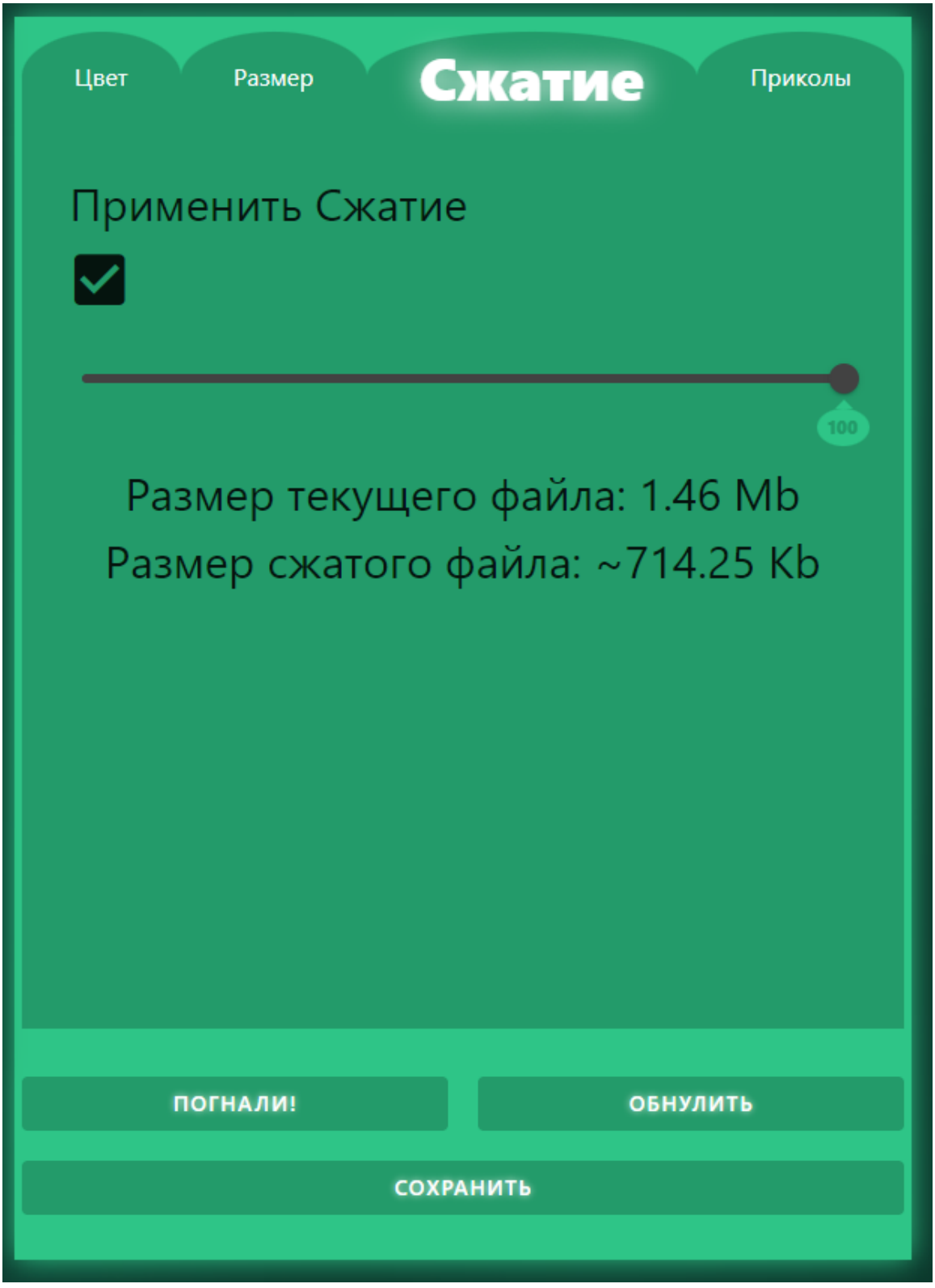


Рисунок 8. Панель сжатия

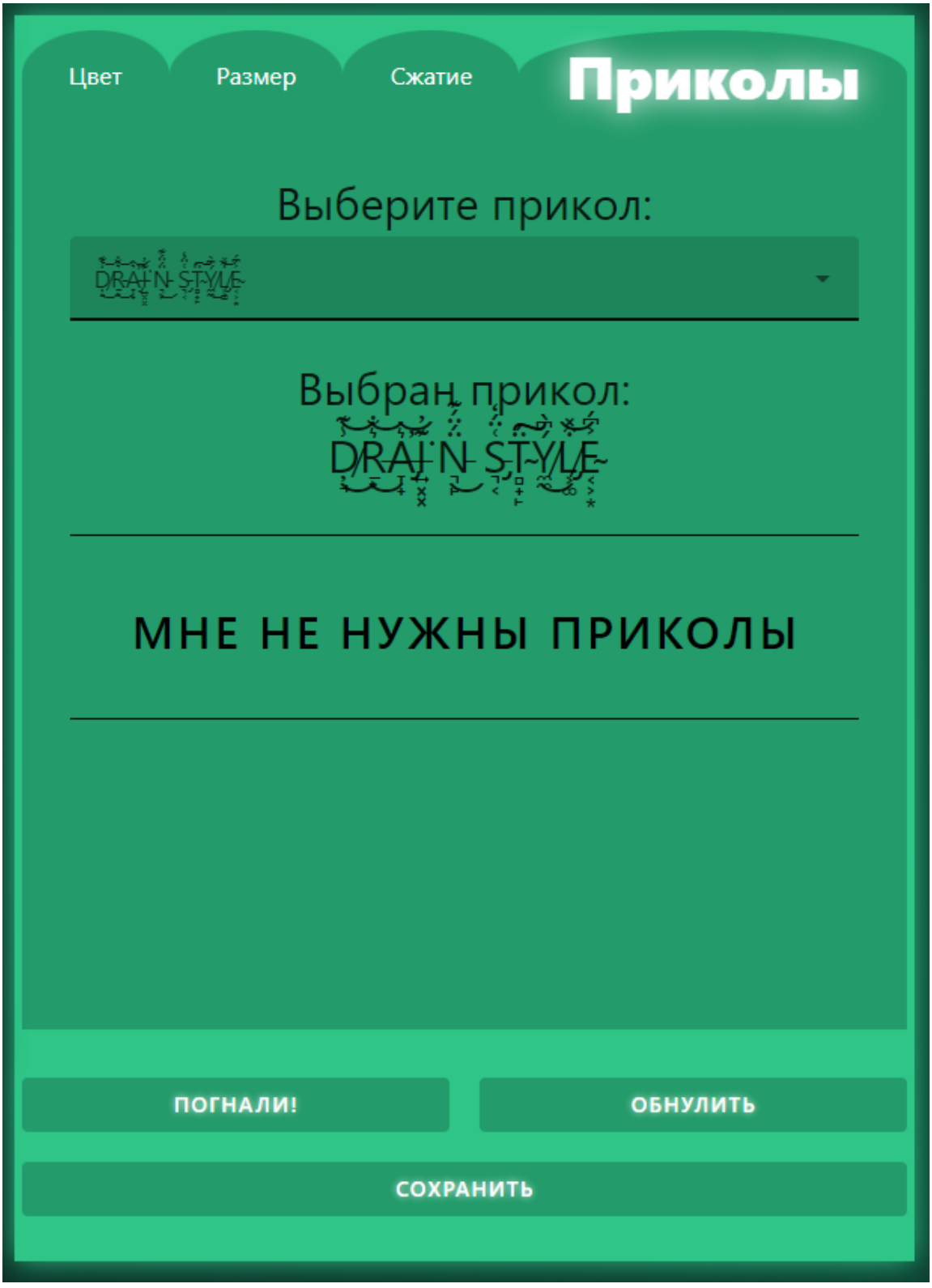


Рисунок 9. Панель произвольных эффектов

После применения различных параметров получилось 6 слоёв:

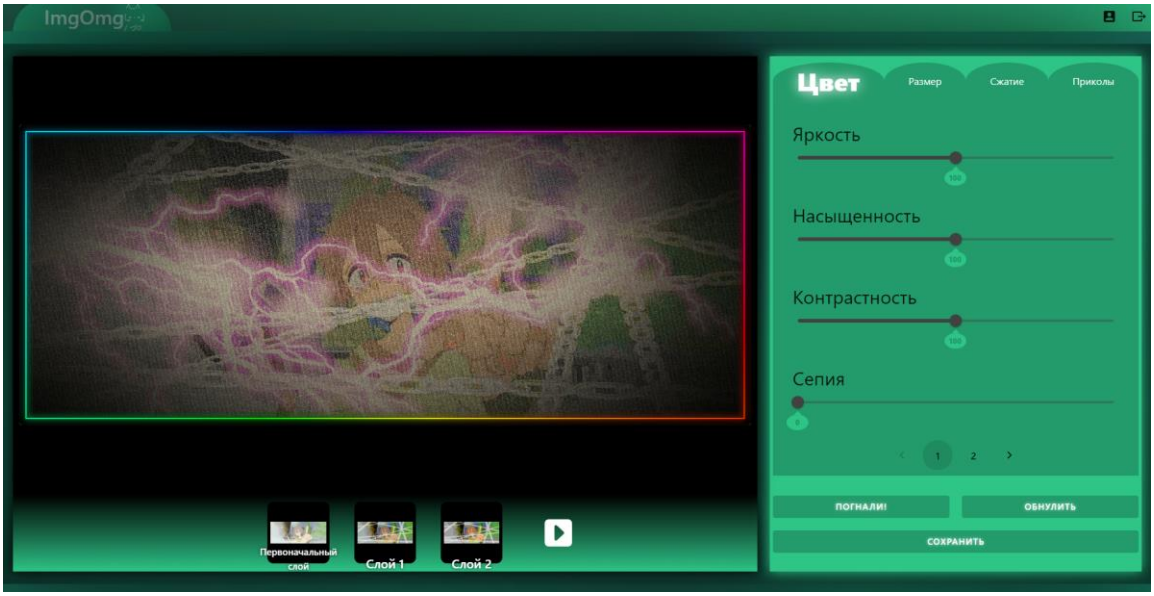


Рисунок 10. Полученное изображение с первыми 3 слоями

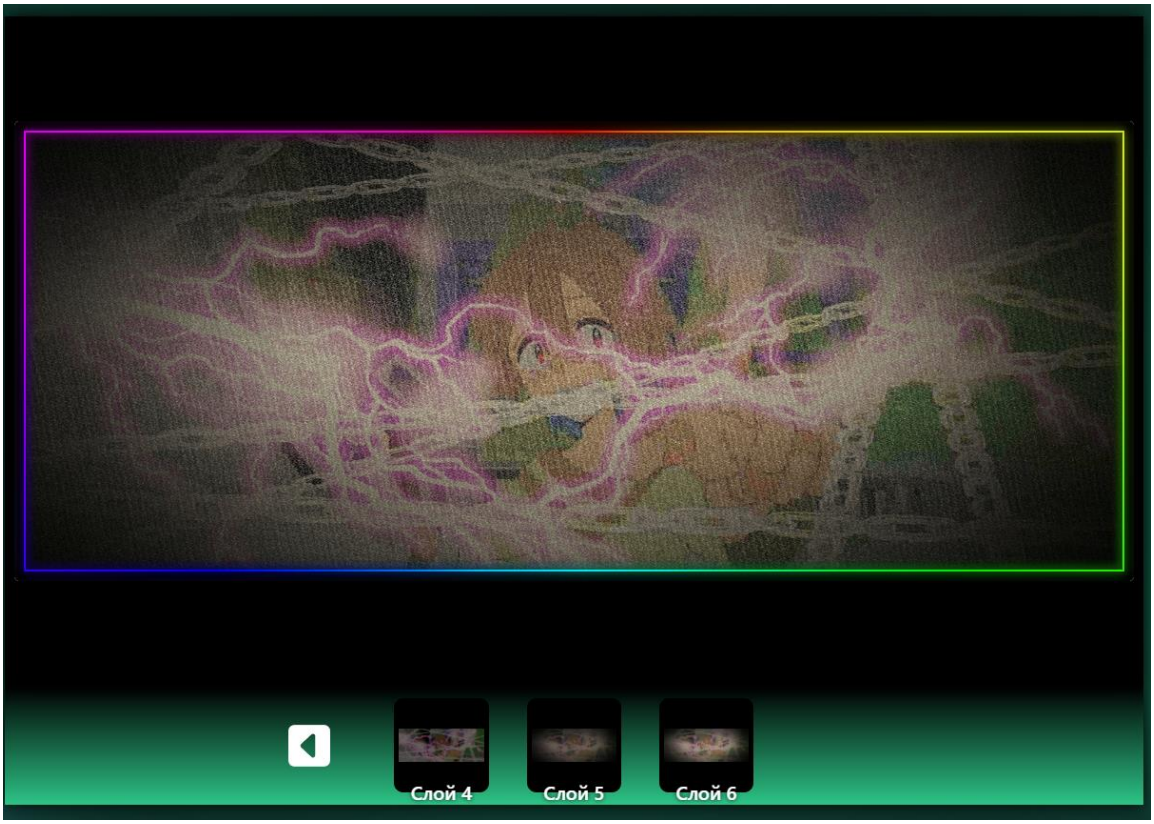


Рисунок 11. Полученное изображение с последними тремя слоями

Можно перейти на любой из слоёв и сохранить изображение на устройство.

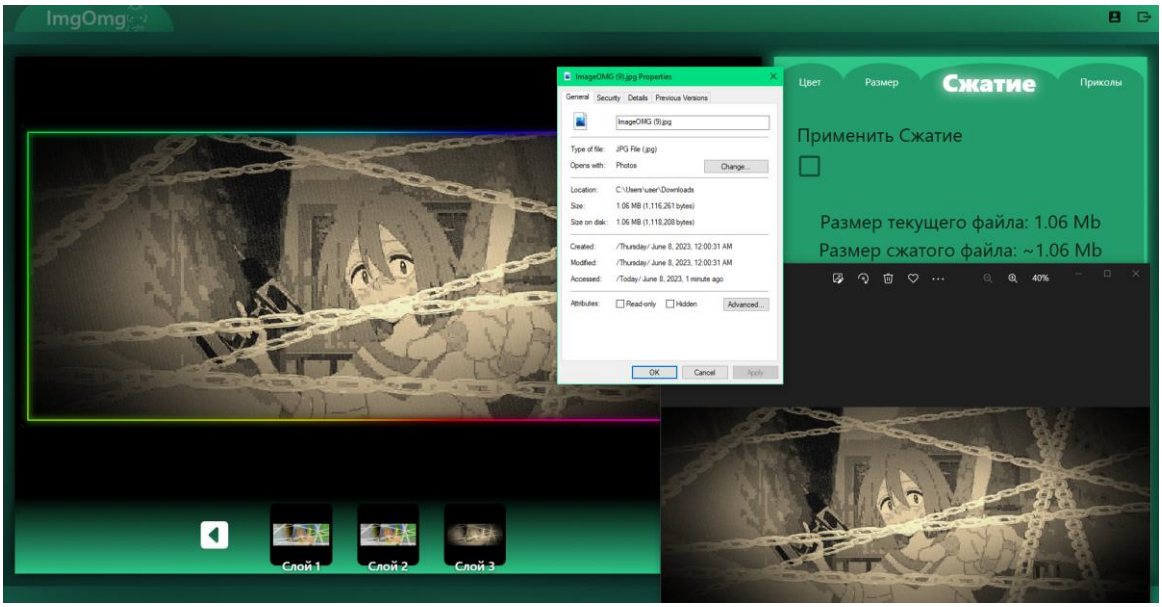


Рисунок 12. Изображение сохранилось и её размер соответствует сжатию

— “/lk”. Личный кабинет выглядит следующим образом:

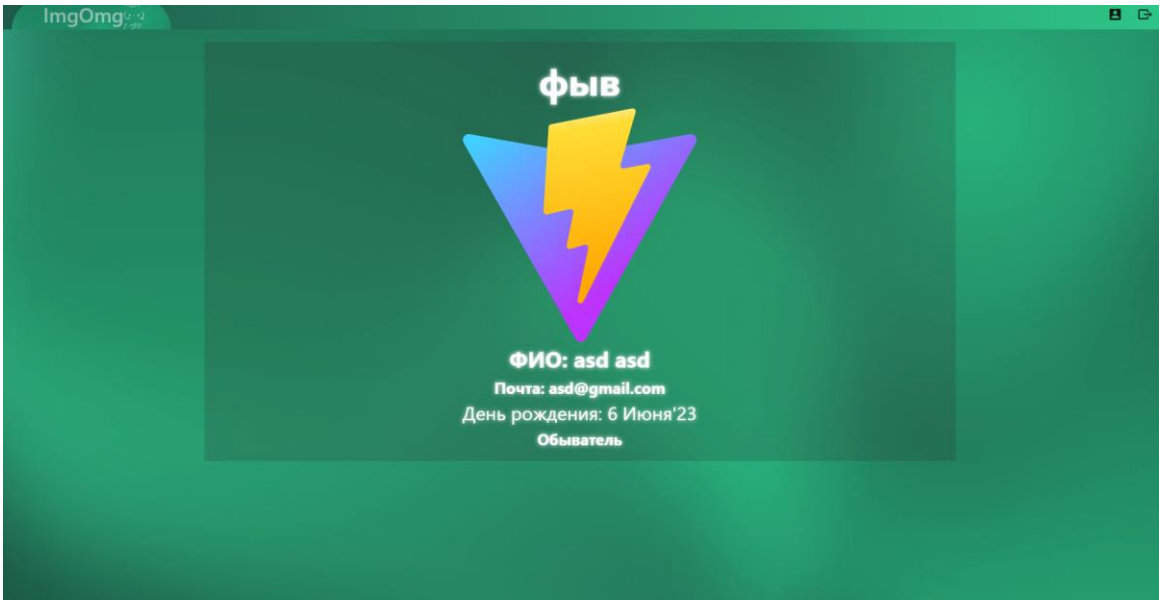


Рисунок 13. Личный кабинет

В самом верху находится логин, ниже фотография пользователя (по умолчанию иконка Vite), ФИО, почта, дата рождения и категория пользователя. Если пользователь выйдет, находясь в личном кабинете, то вид изменится на следующий:

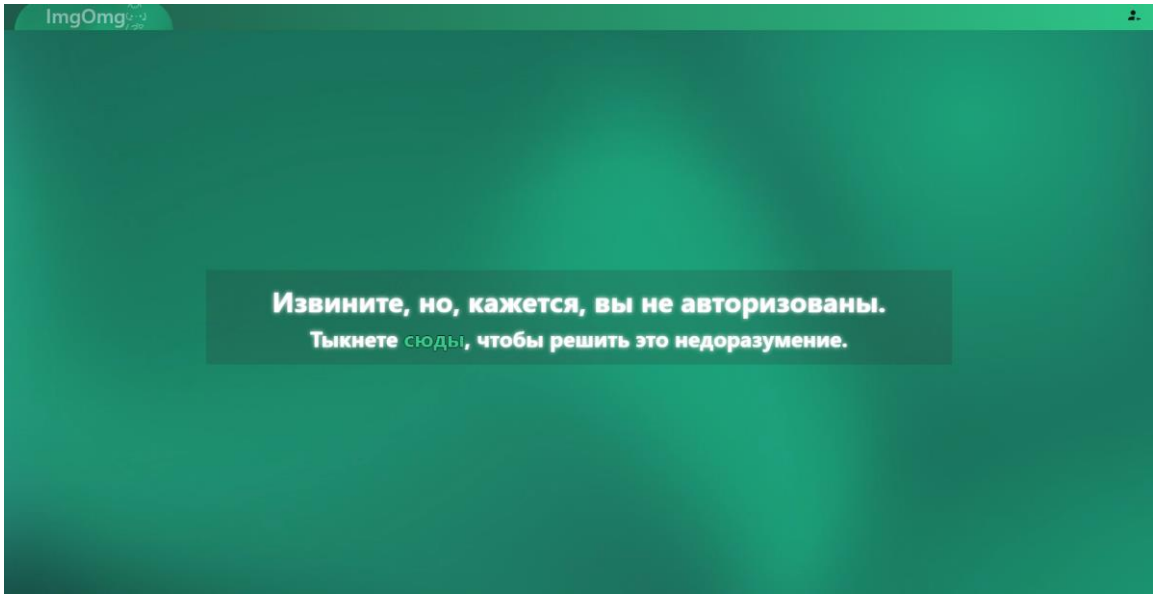


Рисунок 14. Личный кабинет при выходе из учётной записи

4.3.2 Подробное рассмотрение структуры проекта

Всё начинается в “index.html”, который содержит в себе глобальную таблицу стилей и “main.js”:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />
    <link rel="stylesheet" href="./src/style.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>ImgOmg</title>
</head>
```

```
<body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
</body>
</html>
```

“main.js” содержит в себе код для создания приложения на Vue 3 и добавления различных сторонних компонентов:

— Иконки из “Font Awesome 5”

— Выбор даты “Vue Date Picker”

— Хранилище “Vuex”

— UI библиотека “Vuetify 3”

— Роутинг “Router”

Миксины:

— “imageMixin.js” для подсчёта размера изображения

— “profileMixin.js” для составления ФИО и даты.

Множество компонентов:

— Компоненты из папки “editor”, представляющие из себя панели редактирования цвета, размеров, сжатия и приколов.

— “LayerPhoto.vue” – иконка со слоем.

— “NavBar.vue” – навигационная панель сверху экрана.

— “Register.vue” и “Auth.vue” – компоненты панели регистрации и авторизации соответственно.

— “DnlkkDialog.vue” – компонент всплывающего окна, переключающий панель входа/регистрации.

— “DnlkkHomeButton.vue” – большая квадратная кнопка по центру.

— “DnlkkImgOmgPanel.vue” – панель редактирования изображений. Включает в себя все элементы, которые служат для редактирования изображений.
— “DnlkkIcon.vue” – название проекта слева на навигационной панели.
— “CabinetPage.vue” и “HomePage.vue” – главные компоненты для “/lk” и “/” соответственно.
— “App.vue” – самый главный компонент в любом Vue.js приложении.

4.4 База данных

Состоит из 2 таблиц:

— “users” – хранит в себе пользователей с id, логином, паролем, именем, фамилией, отчеством, почтой, днём рождения и id категории
— “categories” – хранит в себе категории пользователей с id, названием категории. Категории определены сразу при инициализации и не изменяются: Админ, Обыватель, Студент, Дизайнер.

4.5 Nginx

Количество рабочих процессов = 4, максимальное количество одновременных соединений рабочего процесса = 1024, слушает порт 80, максимальный размер запроса 20мб, имеет 3 локации:

— “/” – проксирует запросы на frontend на порте 5173.
— “/api/editor” – проксирует запросы на микросервис редактора изображений на порте 5001.

— “/api/auth” – проксирует запросы на микросервис регистрации/авторизации/аутентификации на порте 5002.

Листинг кода “nginx.conf”:

```
worker_processes 4;
events { worker_connections 1024; }
http {
  client_max_body_size 20m;

  server {
    listen 80;
    listen [::]:80;
    server_name localhost;



    location / {
      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_pass http://frontend:5173;
    }

    location /api/editor {
      proxy_pass http://editor:5001;
    }

    location /api/auth {
      proxy_pass http://auth:5002;
    }
  }
}
```

4.6 Docker

В “docker-compose.yml” описывается 5 сервисов:

— “frontend” – билдит Dockerfile фронтенда в папке “./frontend”, имя этого контейнера и остальных равно названию сервиса. Сеть: “frontend_nw”.
— “auth” – тоже, что и прошлое, но для микросервиса регистрации/авторизации/аутентификации.
— “editor” – тоже, что и прошлые, но для микросервиса редактора изображений.
— “database” – запускается на образе “postgres:10.0” на порте 5432. Использует файл окружения из папки “./db/database.env”. Использует том “./db/init” для инициализации бд и том “./db/data” для хранения состояния бд. “Healthcheck“ проверяет работоспособность бд и пытается восстановить её, если она упала. Перезапускается при ошибке. Находится в сетях backend микросервисов.
— “nginx” – билдит Dockerfile из папки “./nginx”. Работает на порте 80. Определяет зависимости от frontend и backend сервисов. Находится во всех сетях.
— Далее определяются сети для frontend и двух backend сервисов.

Листинг “docker-compose.yml”:

```
version: '3.9'

services:
  frontend:
    build: ./frontend
    container_name: frontend
    networks:
      - frontend_nw
```

```
auth:
  build: ./backend/auth
  container_name: auth
  networks:
    - auth_nw

editor:
  build: ./backend/editor
  container_name: editor
  networks:
    - editor_nw

database:
  image: "postgres:10.0"
  ports:
    - 5432:5432
  env_file:
    - ./db/database.env
  volumes:
    - ./db/init:/docker-entrypoint-initdb.d
    - ./db/data:/var/lib/postgresql/data
  healthcheck:
    test: [ "CMD", "pg_isready", "-U", "postgres" ]
    interval: 5s
    retries: 5
  restart: on-failure
  networks:
    - auth_nw
    - editor_nw

nginx:
  build: ./nginx
  ports:
    - 80:80
  depends_on:
```

```
      - frontend
      - auth
      - editor
    restart: always
    networks:
      - frontend_nw
      - auth_nw
      - editor_nw

networks:
  frontend_nw:
  auth_nw:
  editor_nw:
```

Рассмотрим Dockerfile для каждого сервиса (за исключением бд по понятным причинам):

— Для “auth”. Основывается на образе Python 3.11. Устанавливается рабочая папка “/app” и копируется всё содержимое в неё. Далее устанавливаются зависимости из файла “./requirements.txt”. Слушается порт 5002 и запускается “main.py” с нужными параметрами, рассмотренными ранее.

Листинг кода:

```
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install -r ./requirements.txt
EXPOSE 5002
CMD ["python", "main.py", "database", "danilkk", "123", "danilkk_db", "5002"]
```

— Для “editor”. Первые три пункта такие же, как и в прошлом. Далее устанавливается библиотека для работы OpenCV и зависимости из файла “./requirements.txt”. Слушается порт 5001 и запускается “main.py” с нужными параметрами.

Листинг кода:

```
FROM python:3.11
WORKDIR /app
COPY . .
RUN apt-get update \
    && apt-get install -y libgl1-mesa-glx \
    && pip install -r ./requirements.txt
EXPOSE 5001
CMD ["python", "main.py", "5001"]
```

— Для “frontend”. За основу берётся образ “node:latest”. Создаётся рабочая папка “/app”, копируется всё содержимое и устанавливаются зависимости из “package.json” при помощи команды “npm install”. Далее слушается порт 5173 и запускается frontend в режиме разработки с параметром “—host 0.0.0.0”.

Листинг кода:

```
FROM node:latest
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 5173
CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

— Для “nginx”. За основу берётся образ Alpine Linux. Запускает множество команд по установке nginx, созданию символической ссылки, перенаправляющей вывод журнала доступа и ошибок, директории для хранения конфигов сайтов, временных файлов Nginx, удаляет все конфигурации nginx, очищает кеш пакетного менеджера и создаёт рабочую директорию “/app”. Далее копируется конфигурация nginx в папки “./etc/nginx” и “./etc/nginx/conf.d”. Слушает порты 80 и 443. И последнее:

запускает nginx и отключает режим демона, чтобы процесс работал на переднем плане.

Листинг кода:

```
FROM alpine:latest
RUN apk --update add nginx && \
   ln -sf /dev/stdout /var/log/nginx/access.log && \
   ln -sf /dev/stderr /var/log/nginx/error.log && \
   mkdir /etc/nginx/sites-enabled/ && \
   mkdir -p /run/nginx && \
   rm -fr /etc/nginx/conf.d/* &&\
    rm -rf /etc/nginx/nginx.conf &&\
   rm -rf /var/cache/apk/* &&\
   mkdir /app
COPY ./nginx.conf ./etc/nginx/conf.d/default.conf
COPY ./nginx.conf ./etc/nginx/nginx.conf
EXPOSE 80 433
CMD ["nginx", "-g", "daemon off;"]
```

5 Задачи, решённые в ходе выполнения задания

В ходе выполнения задания были решены следующие задачи:

- Создано полноценное REST API приложение, где реализованы клиентская и серверная части, настроено проксирование и все сервисы развёрнуты в контейнерах и объединены с помощью Docker Compose.
- Изучено два фреймворка для серверной части на python.
- Спроектированы архитектуры для микросервисов на python с учётом знаний, полученных в курсе “Языки и системы программирования”.
- Выполнены все требования проекта, поставленные в условиях задания.