

# **САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Математико-механический факультет**

**Кафедра системного программирования**

## **Разработка адаптивного веб-приложения для восстановления мелкой моторики рук у пациентов в период реабилитации**

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**Марахова Екатерина Игоревна**

**2 курс, группа 23.M07-мм**

Научный руководитель:  
доц. кафедры Системного  
программирования  
Луцив Д.В.

Санкт-Петербург, 2025

## Оглавление

<b>ВВЕДЕНИЕ.....</b>	<b>4</b>
Цель и задачи работы .....	4
Научная новизна и практическая значимость .....	5
<b>ГЛАВА 1. АНАЛИЗ СОВРЕМЕННЫХ МЕТОДОВ ВОССТАНОВЛЕНИЯ МЕЛКОЙ МОТОРИКИ РУК .....</b>	<b>6</b>
1.1. Современные подходы к реабилитации мелкой моторики .....	6
1.2. Обзор существующих решений: аппаратные и программные методы.....	6
1.3. Обзор существующих моделей компьютерного зрения для распознавания жестов рук .....	8
1.4. Обоснование выбора программного подхода .....	9
<b>ГЛАВА 2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ</b>	<b>11</b>
2.1. Анализ требований к системе .....	11
2.2. Сравнительный анализ архитектурных подходов .....	11
2.3. Обоснование выбора технологического стека .....	12
2.4. Организация межсервисного взаимодействия .....	13
2.5. Модель данных и проектирование API.....	13
<b>ГЛАВА 3. РЕАЛИЗАЦИЯ МИКРОСЕРВИСОВ .....</b>	<b>15</b>
3.1. Сервис аутентификации и авторизации .....	15
3.1.1. Регистрация, вход и восстановление пароля .....	15
3.1.2. JWT-аутентификация .....	16
3.1.3. Интеграция с фронтендом (React).....	17
3.2. Сервис выполнения упражнений .....	19
3.2.1. Работа с MediaPipe: распознавание жестов.....	19

3.2.2. Процесс получения изображения с камеры и формирования упражнений .....	19
3.2.3. Алгоритм генерации упражнений.....	21
3.2.4. Примеры конфигураций.....	22
3.2.5. Особенности реализации .....	23
<b>3.3. Разработка пользовательского интерфейса .....</b>	<b>24</b>
3.3.1. UX/UI-дизайн для пациентов .....	24
3.3.2. Взаимодействие с серверной частью .....	25
<b>3.4. Контейнеризация приложения .....</b>	<b>25</b>
3.4.1. Контейнеризация сервисов (Docker).....	25
3.4.2. Настройка docker-compose.yml.....	26
<b><i>ЗАКЛЮЧЕНИЕ</i> .....</b>	<b>28</b>
<b>Результаты работы.....</b>	<b>28</b>
<b>Перспективы развития проекта .....</b>	<b>28</b>
<b><i>СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ</i>.....</b>	<b>30</b>

## **ВВЕДЕНИЕ**

Восстановление мелкой моторики рук является важной составляющей реабилитации пациентов после инсультов, травм, неврологических заболеваний и операций. Традиционные методы реабилитации, такие как механические тренажеры и экзоскелеты, требуют значительных финансовых затрат, индивидуальной настройки и зачастую недоступны для широкого круга пациентов. В то же время развитие компьютерных технологий и машинного обучения открывает новые возможности для создания программных решений, способных обеспечить эффективную и доступную реабилитацию.

Адаптивные веб-приложения, использующие алгоритмы компьютерного зрения для анализа движений, позволяют проводить реабилитационные упражнения в домашних условиях без дорогостоящего оборудования. Однако существующие аналоги либо обладают ограниченным функционалом, либо являются коммерческими продуктами с высокой стоимостью использования. В связи с этим разработка бесплатного, удобного и технологичного решения для восстановления мелкой моторики представляется актуальной задачей, способной улучшить качество реабилитационного процесса для широкого круга пациентов.

### **Цель и задачи работы**

Целью данной работы является разработка адаптивного веб-приложения с микросервисной архитектурой для восстановления мелкой моторики рук, использующего технологии компьютерного зрения.

Для достижения поставленной цели решаются следующие задачи:

1. провести анализ современных методов реабилитации мелкой моторики и существующих программных решений;
2. спроектировать микросервисную архитектуру приложения, обеспечивающую модульность и масштабируемость;

3. реализовать сервис аутентификации и авторизации пользователей на основе Django и JWT;
4. разработать сервис выполнения упражнений с использованием библиотеки MediaPipe для распознавания жестов;
5. создать интуитивно понятный пользовательский интерфейс на React, адаптированный для пациентов с ограниченной моторикой;
6. обеспечить развертывание приложения с помощью Docker для упрощения его распространения и использования.

### **Научная новизна и практическая значимость**

Научная новизна работы заключается в применении микросервисного подхода к разработке реабилитационного веб-приложения, а также в интеграции алгоритмов компьютерного зрения (MediaPipe) для точного распознавания жестов без использования специализированного оборудования. В отличие от существующих решений, предлагаемая система сочетает доступность, модульность и адаптивность, что расширяет возможности ее применения в медицинской и реабилитационной практике.

Практическая значимость исследования состоит в создании бесплатного и удобного инструмента для пациентов, проходящих реабилитацию. Разработанное приложение позволяет проводить упражнения в любых условиях, используя стандартную веб-камеру, что снижает финансовые барьеры и повышает доступность восстановительных методик. Кроме того, использование Docker обеспечивает простоту развертывания системы, что делает ее применимой в медицинских учреждениях и для домашнего использования.

Результаты работы могут быть полезны для реабилитологов, пациентов с нарушениями моторики, а также разработчиков медицинских программных решений, заинтересованных в создании современных цифровых инструментов для восстановительной медицины.

# **ГЛАВА 1. АНАЛИЗ СОВРЕМЕННЫХ МЕТОДОВ ВОССТАНОВЛЕНИЯ МЕЛКОЙ МОТОРИКИ РУК**

## **1.1. Современные подходы к реабилитации мелкой моторики**

Восстановление мелкой моторики рук является важной составляющей реабилитации пациентов с последствиями инсультов, черепно-мозговых травм, повреждений периферических нервов и нейродегенеративных заболеваний. Современные подходы к реабилитации можно условно разделить на три основные группы.

1. Традиционные методы (лечебная физкультура, мануальная терапия, эрготерапия) – основаны на повторяющихся упражнениях под контролем специалиста [1].
2. Аппаратные методы (механические тренажеры, роботизированные системы, экзоскелеты) – обеспечивают пассивную и активную разработку суставов с помощью технических устройств.
3. Программно-аппаратные и цифровые решения (компьютерные тренажеры, VR/AR-системы, приложения с компьютерным зрением) – используют интерактивные технологии для мотивации пациента и объективной оценки прогресса.

В последние годы наблюдается тенденция к цифровизации реабилитационных методик, что позволяет повысить их доступность и эффективность за счет автоматизации контроля выполнения упражнений и адаптивного подхода к нагрузкам.

## **1.2. Обзор существующих решений: аппаратные и программные методы**

### **Аппаратные решения**

Среди российских разработок в области аппаратной реабилитации мелкой моторики можно выделить следующие.

- Экзоскелеты (например, «Аника») – механические устройства, фиксирующие пальцы и запястье, обеспечивающие дозированную нагрузку. Их главный недостаток – высокая стоимость (от 300 тыс. руб.) и необходимость индивидуальной настройки [2].
- Роботизированные тренажеры (например, «Армед») – устройства с электроприводом, помогающие выполнять сгибание-разгибание пальцев. Требуют подключения к стационарным системам и контроля со стороны врача.
- Механические тренажеры (например, «Бутон») – простые устройства для разработки хватательных движений. Имеют ограниченную функциональность и не обеспечивают обратную связь.

### **Программные решения**

В отличие от аппаратных методов, цифровые технологии предлагают более гибкие и доступные варианты реабилитации.

- VR-тренажеры (например, «Мотива VR») – используют виртуальную реальность для игровой реабилитации, но требуют дорогостоящего оборудования.
- Мобильные приложения (например, «Rehand») – предлагают упражнения с тачскрином, но не анализируют правильность выполнения движений.
- Компьютерные системы с камерой (например, Kinect-реабилитация) – позволяют отслеживать движения без дополнительных датчиков, но имеют ограниченную точность.

Наиболее перспективными представляются гибридные решения, сочетающие программные методы с минимальным использованием аппаратных средств (например, веб-камера).

### 1.3. Обзор существующих моделей компьютерного зрения для распознавания жестов рук

Современные подходы к распознаванию жестов рук можно разделить на три основные категории.

1. Классические алгоритмы компьютерного зрения (например, на основе OpenCV) :
  - используют детекцию ключевых точек через фильтры (SIFT, SURF);
  - требуют сложной предобработки изображений;
  - чувствительны к освещению и фону.
2. Традиционные нейросетевые архитектуры (CNN, LSTM)
  - эффективны для статичных жестов;
  - требуют больших размеченных датасетов;
  - пример: HSE GestureNet (разработка ВШЭ, 2021) – компактная CNN для распознавания 20 жестов с точностью ~89% [3].
3. Современные end-to-end решения (MediaPipe, MMPose);
  - комбинируют детекцию и классификацию;
  - работают в реальном времени;
  - оптимизированы для мобильных и веб-приложений.

Рассмотренные ИИ-модели для детекции рук также отображены в сравнительной таблице 1.

«Таблица 1. Сравнение популярных моделей»

Модель	Точность	FPS	Особенности
MediaPipe Hands	95% (21 точка)	30+	Готовая JS-реализация, не требует серверной обработки
MMPose	93%	25	Поддержка 3D-реконструкции
OpenPose	91% (21 точка)	10	Тяжелая модель, требует GPU
HSE GestureNet	89% (20 жестов)	15	Компактный размер (5 МБ), требует дообучения



Для проекта была выбрана библиотека MediaPipe Hands по следующим причинам.

- Готовая интеграция с JavaScript через CDN (`@mediapipe/hands`).
- Высокая производительность даже на слабых устройствах.
- Отсутствие серверной нагрузки – обработка происходит на клиенте.
- Поддержка нескольких рук одновременно.
- Кросс-платформенность (работает в любом современном браузере).
- Библиотека MediaPipe обеспечивает высокую точность распознавания жестов в реальном времени [4].

Альтернатива в виде HSE GestureNet потребовала бы:

- дополнительного серверного API для инференса.
- создания датасета под конкретные реабилитационные упражнения.
- оптимизации для работы в реальном времени.

Таким образом, MediaPipe оптимально сочетает точность, производительность и простоту интеграции в веб-приложение.

#### **1.4. Обоснование выбора программного подхода**

Анализ существующих решений показывает, что аппаратные методы, несмотря на высокую точность, остаются малодоступными из-за стоимости и сложности эксплуатации. VR-системы также требуют значительных затрат на оборудование, а мобильные приложения не обеспечивают достаточной обратной связи.

В связи с этим программный подход на основе веб-технологий и компьютерного зрения (MediaPipe) представляется оптимальным по следующим причинам.

1. Доступность – не требует специализированного оборудования, только ПК/ноутбук с камерой.
2. Масштабируемость – веб-приложение может использоваться одновременно множеством пациентов.

3. Точность – современные библиотеки (MediaPipe, OpenCV) позволяют анализировать движения с достаточной для реабилитации точностью.
4. Адаптивность – упражнения в будущем можно будет корректировать в зависимости от прогресса пациента.

## **ГЛАВА 2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ**

### **2.1. Анализ требований к системе**

Разработка адаптивного веб-приложения для реабилитации мелкой моторики рук предъявляет комплекс требований к системе, которые можно разделить на функциональные и нефункциональные. Ключевым функциональным требованием является обеспечение точного распознавания жестов и движений кисти в реальном времени с использованием исключительно встроенных возможностей веб-браузера. Это подразумевает реализацию механизма захвата видеопотока с веб-камеры средствами JavaScript без применения дополнительных библиотек компьютерного зрения, таких как OpenCV.

Важным аспектом является обеспечение модульности системы, что позволяет выделить сервис аутентификации и сервис упражнений в независимые компоненты. Такое разделение обусловлено различной природой этих функциональных блоков и необходимостью их независимого масштабирования. Сервис аутентификации должен обеспечивать безопасное хранение пользовательских данных и поддерживать механизм JWT-авторизации, в то время как сервис упражнений отвечает за обработку данных о выполнении реабилитационных заданий.

### **2.2. Сравнительный анализ архитектурных подходов**

При проектировании системы рассматривались две принципиально разные архитектурные парадигмы: монолитная и микросервисная. Монолитная архитектура, при всей своей простоте реализации и развертывания, демонстрирует существенные ограничения в контексте разрабатываемого приложения. Основным недостатком монолитного подхода является высокая степень связанности компонентов, что затрудняет внесение

изменений в отдельные функциональные модули и ограничивает возможности масштабирования системы.

Микросервисная архитектура, напротив, предоставляет значительные преимущества для данного проекта. Разделение системы на независимые сервисы позволяет оптимизировать нагрузку на отдельные компоненты и обеспечивает гибкость при дальнейшем расширении функциональности. В частности, выделение сервиса аутентификации в отдельный модуль повышает безопасность системы и упрощает реализацию механизмов авторизации. При этом важно отметить, что выбор микросервисной архитектуры потребовал решения дополнительных задач, связанных с организацией межсервисного взаимодействия и обеспечением согласованности данных. Преимущества микросервисной архитектуры для медицинских систем подробно описаны в работе [5].

### **2.3. Обоснование выбора технологического стека**

Технологический стек проекта был подобран с учетом специфических требований к системе. Для реализации серверной части был выбран фреймворк Django в сочетании с Django REST Framework. Этот выбор обусловлен наличием встроенной системы аутентификации, поддержкой ORM для работы с базой данных и высокой производительностью при обработке API-запросов. В качестве клиентского фреймворка был выбран React, который благодаря компонентной архитектуре и эффективному механизму виртуального DOM обеспечивает необходимую интерактивность пользовательского интерфейса.

Особое внимание было уделено выбору технологии для распознавания жестов. Библиотека MediaPipe Hands, доступная через CDN (<https://cdn.jsdelivr.net/npm/@mediapipe/hands/>), была выбрана благодаря своей способности работать непосредственно в браузере без необходимости серверной обработки видеопотока. Это решение обеспечивает распознавание 21 ключевой точки на кисти с частотой до 30 кадров в секунду, что полностью

удовлетворяет требованиям реабилитационных упражнений. Для контейнеризации сервисов был выбран Docker, который обеспечивает изоляцию компонентов и упрощает процесс развертывания системы.

## **2.4. Организация межсервисного взаимодействия**

Взаимодействие между компонентами системы организовано через REST API с использованием JSON-формата данных. Сервис аутентификации предоставляет эндпоинты для регистрации и авторизации пользователей, возвращая JWT-токены для последующей аутентификации запросов. Механизм JWT был выбран благодаря его статистической природе, что позволяет избежать необходимости хранения состояния сессии на сервере.

Клиентское приложение, построенное на React, взаимодействует с сервисом упражнений через защищенные HTTPS-соединения. Особенностью архитектуры является то, что обработка видеопотока и распознавание жестов полностью осуществляется на клиентской стороне с использованием MediaPipe Hands, что значительно снижает нагрузку на серверную часть. Данные о выполнении упражнений передаются на сервер только после завершения сеанса реабилитации, что оптимизирует использование сетевых ресурсов.

## **2.5. Модель данных и проектирование API**

На данный момент модель данных системы сущность пользователя. Дальнейшая разработка предполагает включение трех основных сущностей: пользователя, упражнение и сессию выполнения упражнения. Пользовательская сущность содержит информацию для аутентификации и персональные данные. Сущность упражнения описывает параметры реабилитационных заданий, включая степень сложности и целевую группу мышц. Сущность сессии фиксирует результаты выполнения упражнения конкретным пользователем.

API системы спроектировано в соответствии с принципами REST и включает набор эндпоинтов для работы с каждой сущностью. Для обеспечения безопасности все запросы, кроме операций регистрации и аутентификации, требуют наличия валидного JWT-токена в заголовке Authorization. Особенностью API является его ориентация на работу с клиентским приложением, что выражается в оптимизированных ответах, содержащих только необходимые для отображения данные [6].

## ГЛАВА 3. РЕАЛИЗАЦИЯ МИКРОСЕРВИСОВ

### 3.1. Сервис аутентификации и авторизации

Сервис аутентификации представляет собой ключевой компонент системы, обеспечивающий безопасное управление пользовательскими данными. Реализация выполнена на базе Django REST Framework с использованием JWT (JSON Web Tokens) в качестве основного механизма аутентификации. Архитектура сервиса построена по принципу RESTful API, что обеспечивает его совместимость с различными клиентскими приложениями.

#### 3.1.1. Регистрация, вход и восстановление пароля

Процесс регистрации пользователя реализован через эндпоинт `/api/auth/register`, принимающий обязательные параметры: электронную почту и пароль. При валидации данных система проверяет уникальность email и соответствие пароля требованиям сложности (минимум 8 символов, включая цифры и специальные знаки). После успешной регистрации создается запись в базе данных SQLite в зашифрованном виде с использованием алгоритма PBKDF2 с SHA-256 с хэшированием, что обеспечивает необходимый уровень безопасности.

Для входа в систему используется эндпоинт `/api/auth/login`, который при корректных учетных данных возвращает пару токенов:

- Access Token (срок действия 15 минут);
- Refresh Token (срок действия 24 часа).

Механизм восстановления пароля включает:

1. генерацию уникального токена с привязкой к пользователю;
2. отправку письма с ссылкой для сброса через SMTP-сервер (например, SendGrid);
3. валидацию токена при переходе по ссылке;

#### 4. обновление пароля через эндпоинт /api/auth/reset-password.

```
class PasswordResetRequestView(APIView):
    @swagger_auto_schema(
        request_body=PasswordResetSerializer,
        operation_description="Сброс пароля (отправка email)",
        responses={200: 'Если email зарегистрирован, письмо отправлено.'}
    )
    def post(self, request):
        load_dotenv()
        email = request.data.get('email')
        user = User.objects.filter(email=email).first()
        if user:
            uid = urlsafe_base64_encode(force_bytes(user.pk))
            token = default_token_generator.make_token(user)
            reset_url = f"http://localhost:3000/password-reset-confirm/{uid}/{token}/"
            send_mail(
                subject='Сброс пароля',
                message=f'Перейдите по ссылке для сброса пароля: {reset_url}',
                os.getenv('EMAIL_HOST_USER'),
                recipient_list=[email],
            )
        return Response(data={'message': 'Если email зарегистрирован, письмо отправлено.'}, status=status.HTTP_200_OK)
```

Рис.1 Пример кода Django для отправки письма с токеном

### 3.1.2. JWT-аутентификация

Система использует библиотеку Simple JWT, которая реализована с использованием стандартных возможностей DRF [7]:

- подпись токенов по алгоритму HS256;
- автоматическую проверку срока действия;
- механизм обновления Access Token через Refresh Token.

Схема работы сервиса (см. Рис. 2):

1. клиент отправляет credentials (email + пароль);
2. сервер проверяет их и выдает токены;
3. все последующие запросы содержат Access Token в заголовке Authorization: Bearer <token>;
4. при истечении срока Access Token клиент запрашивает новый через Refresh Token.



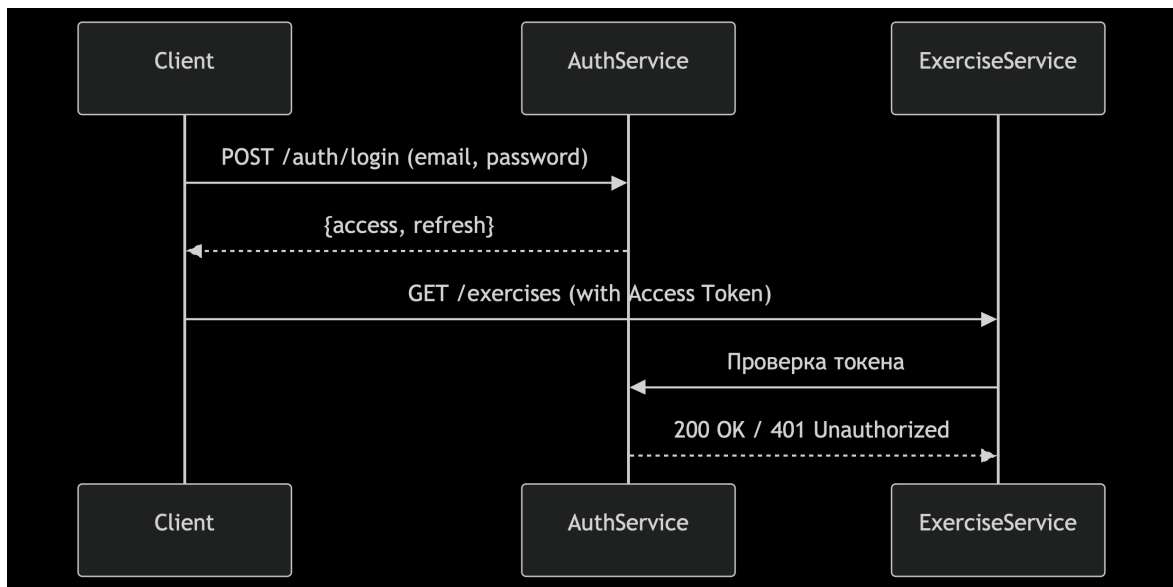


Рис. 2 Схема работы сервиса

### 3.1.3. Интеграция с фронтендом (React)

Интеграция сервиса аутентификации с клиентской частью на React реализована через систему маршрутизации и централизованное управление состоянием аутентификации. Основой интеграции служит компонент App, который выполняет функции корневого элемента приложения и обеспечивает следующие ключевые аспекты взаимодействия.

#### 1. Управление состоянием аутентификации.

Состояние аутентификации контролируется через механизм React-состояния с использованием хука `useState`. Токен доступа сохраняется в локальном хранилище браузера (`localStorage`), что позволяет сохранять сеанс пользователя между перезагрузками страницы. При инициализации приложения выполняется проверка наличия токена в `localStorage`, и его значение устанавливается в начальное состояние компонента App.

#### 2. Система маршрутизации.

Приложение использует библиотеку `react-router-dom` версии 6 для организации навигации. Маршрутизация построена по принципу защищенных маршрутов (`protected routes`), где доступ к определенным страницам (например, `/profile`) ограничен для неавторизованных пользователей.

Реализация защиты маршрутов выполнена через условный рендеринг - если токен отсутствует, пользователь перенаправляется на страницу входа с помощью компонента `Navigate`.

### 3. Обработка выхода из системы.

Функция `handleLogout` обеспечивает завершение сеанса пользователя путем удаления токена из `localStorage` и обновления состояния приложения. Это приводит к автоматическому перенаправлению на страницу входа благодаря механизму защищенных маршрутов.

### 4. Передача параметров между компонентами.

Система реализует однонаправленный поток данных:

- компонент `Login` получает callback-функцию `setToken` для обновления состояния аутентификации;
- компонент `Profile` получает текущий токен и функцию выхода из системы;
- компоненты восстановления пароля работают с динамическими параметрами маршрута (`uid` и `token`).

### 5. Интеграция с UI-библиотеками.

Приложение использует `Bootstrap` для базовых стилей, что обеспечивает согласованный визуальный интерфейс всех компонентов аутентификации. Дополнительные кастомные стили подключаются через файл `index.css`.

Особенностью реализации является отсутствие глобального контекста аутентификации - вместо этого состояние передается явно через `props`. Такой подход упрощает отслеживание потока данных в приложении, хотя и может потребовать модификации при значительном увеличении количества защищенных маршрутов.

Архитектура интеграции демонстрирует следующие преимущества:

- ясное разделение ответственности между компонентами;
- простота отслеживания изменений состояния аутентификации;
- легкость тестирования отдельных компонентов;

- минимальная зависимость от сторонних библиотек управления состоянием.

## **3.2. Сервис выполнения упражнений**

Сервис выполнения упражнений представляет собой ключевой функциональный модуль системы, обеспечивающий распознавание жестов верхних конечностей и контроль за правильностью выполнения реабилитационных упражнений. В основе сервиса лежит оригинальная методика классификации жестов, разработанная специально для задач восстановления мелкой моторики.

### **3.2.1. Работа с MediaPipe: распознавание жестов**

Для распознавания жестов используется библиотека MediaPipe Hands, интегрированная в клиентскую часть приложения через CDN [8]. Особенностью реализации является обработка видеопотока непосредственно в браузере без передачи данных на сервер, что обеспечивает конфиденциальность пользовательской информации и снижает нагрузку на сетевые ресурсы [9].

Библиотека MediaPipe Hands предоставляет 21 ключевую точку для каждой кисти, включая координаты суставов и кончиков пальцев. На основе этих данных разработан алгоритм определения положения пальцев, который учитывает углы между фалангами и их ориентацию относительно ладони. Для повышения точности распознавания введены дополнительные проверки, исключающие ложные срабатывания при частичном перекрытии пальцев или нестандартном положении кисти.

### **3.2.2. Процесс получения изображения с камеры и формирования упражнений**

Обработка видеопотока осуществляется через специальный класс Camera, который инкапсулирует работу с API MediaDevices, что позволяет

получать изображение с веб-камеры пользователя без использования дополнительных библиотек. Хотя прямое обращение к `navigator.mediaDevices.getUserMedia()` не отражено в коде, класс `Camera` выполняет аналогичные функции, обеспечивая:

- инициализацию видеопотока с разрешением 640x480 пикселей;
- регулярный захват кадров через callback-функцию `onFrame`;
- передачу видеок кадров в обработчик `MediaPipe Hands`.

Ключевые особенности реализации:

- видеоэлемент (`videoElement`) служит источником изображения;
- кадры передаются в `MediaPipe` в формате объекта `{image: videoElement}`;
- обработка выполняется асинхронно через `await hands.send()`.

Алгоритм обработки данных включает несколько этапов.

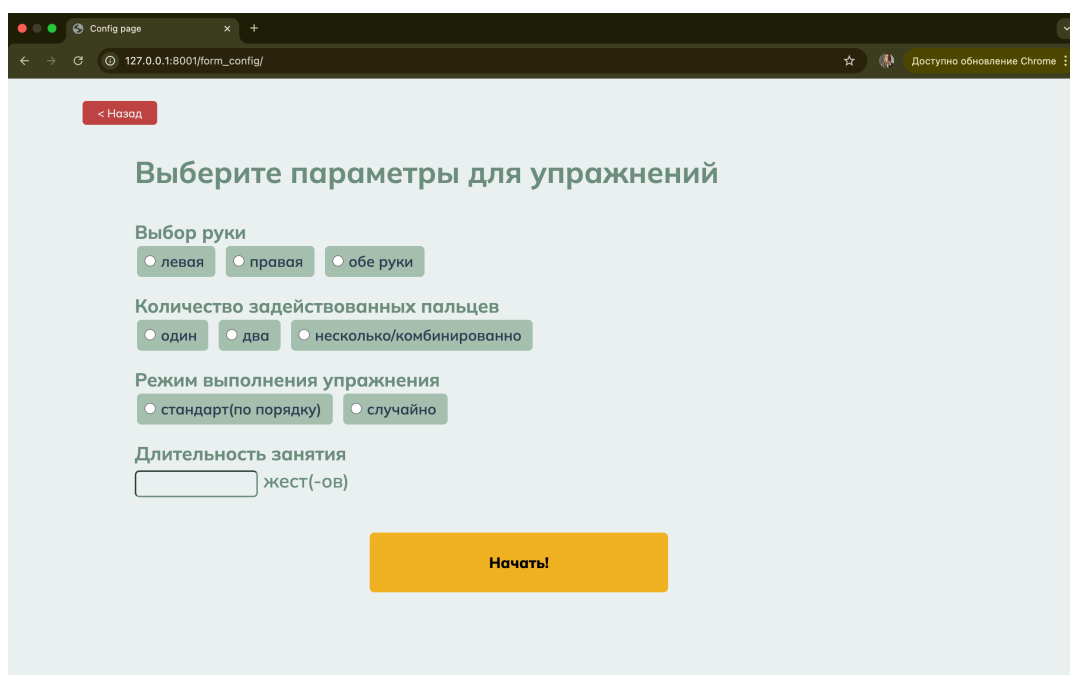
1. Предварительная фильтрация кадров: устранение шумов и нормализация освещенности с помощью алгоритмов бинаризации.
2. Определение ключевых точек: анализ положения суставов с использованием предобученной модели `MediaPipe`.

Классификация распознанного жеста реализована по принципу пятиразрядного числа, где каждый бит соответствует определенному пальцу (от большого к мизинцу). Значение бита "1" указывает на разогнутое положение пальца, "0" - на согнутое. Полученный двоичный код преобразуется в десятичное число, которое служит уникальным идентификатором жеста.

Количество и вариативность упражнений определяется на конфигурационной странице (`configpage.html`, см. рис. 3), которая реализует форму сбора параметров для генерации упражнений через систему радиокнопок и числового ввода. Форма включает четыре ключевых раздела конфигурации:

1. «Выбор руки»:
  - левая рука (значение "Left");

- правая рука (значение "Right");
  - обе руки поочередно (значение "Multi").
2. «Количество пальцев»:
- один палец (значение 1);
  - два пальца (значение 2);
  - комбинированный режим (значение 5).
3. «Режим выполнения»:
- стандартный последовательный (значение "std");
  - случайный порядок (значение "rnd").
4. «Длительность занятия»:
- числовое поле ввода от 1 до 1000 упражнений.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8001/form\_config/'. The page content is in Russian and titled 'Выберите параметры для упражнений'. It contains four sections of configuration options:

- Выбор руки**: Three radio buttons labeled 'левая', 'правая', and 'обе руки'.
- Количество задействованных пальцев**: Three radio buttons labeled 'один', 'два', and 'несколько/комбинированно'.
- Режим выполнения упражнения**: Two radio buttons labeled 'стандарт(по порядку)' and 'случайно'.
- Длительность занятия**: A text input field followed by the text 'жест(-ов)'.

A yellow button labeled 'Начать!' is positioned at the bottom center of the form.

Рисунок 3. Форма конфигурации режима упражнений

### 3.2.3. Алгоритм генерации упражнений

После отправки формы параметры передаются на сервер методом POST и сохраняются в localStorage браузера. На основе этих параметров система формирует массив упражнений (exlist) по следующему сценарию.

Для режима одной руки (Left/Right):

- при выборе 1 пальца используются значения из массива `one_fingers_list` [16, 8, 4, 1, 4, 8, 16];
- для 2 пальцев – `two_fingers_list` [3, 6, 9, 12, 17, 20, 24];
- для 5 пальцев – `more_fingers_list` [16, 24, 28, 31] или `all_fingers` в случайном режиме.

Для режима двух рук (Multi):

- используется базовый набор `fist_and_palm` [0, 31];
- дополнительно формируется `hands_location_list` с чередованием ["Left", "Right"].

Логика заполнения массива:

- в стандартном режиме (std) упражнения повторяются циклически.

```
for(let i=0; i<exercises; i++){
  exlist.push(one_fingers_list[i % one_fingers_list.length]);
}
```

- в случайном режиме (`rnd`) выбираются произвольные элементы.

```
let randomItem = one_fingers_list[Math.floor(Math.random() * one_fingers_list.length)];
exlist.push(randomItem);
```

### 3.2.4. Примеры конфигураций

Далее приведены примеры сформированных списков упражнений.

#### 1. Базовая реабилитация:

- правая рука;
- 1 палец;
- стандартный режим;
- 10 упражнений.

Результат: [16, 8, 4, 1, 4, 8, 16, 8, 4, 1]

#### 2. Продвинутый уровень:

- обе руки;
- комбинированный режим;
- случайный порядок;

– 5 упражнений.

Результат: [31, "Left"], [0, "Right"], [31, "Right"], [0, "Left"], [31, "Left"]

Визуализация процесса представлена на рисунке 4.



Рисунок 4. Визуализация процесса формирования списка упражнений

### 3.2.5. Особенности реализации

Среди особенностей реализации можно выделить следующие:

1. Динамическое обновление интерфейса:
  - изображение-образец меняется при переходе между упражнениями;
  - для левой руки применяется зеркальное отображение.

```
if (hand_location == "Left") {  
    img.style.transform = "scale(-1, 1)";  
}
```

2. Адаптивная обратная связь:
  - подсветка активной руки цветом (#F39C6B для левой, #659B5E для правой);

- отображение счетчика оставшихся упражнений.

### 3. Гибкость системы:

- возможность расширения путем добавления новых массивов жестов;
- простота модификации логики генерации через изменение параметров формы.

Данный подход обеспечивает персонализированную настройку реабилитационных упражнений под индивидуальные потребности пациента, сохраняя при этом простоту интерфейса и гибкость системы.

## 3.3. Разработка пользовательского интерфейса

### 3.3.1. UX/UI-дизайн для пациентов

Пользовательский интерфейс системы разработан с учетом особых потребностей пациентов, проходящих реабилитацию мелкой моторики. В основе дизайна лежат принципы доступности (accessibility) и минимальной когнитивной нагрузки. Цветовая схема построена на контрастных, но не агрессивных тонах, что обеспечивает комфортное восприятие для пользователей с возможными нарушениями зрения. Основные интерактивные элементы имеют размер не менее 48×48 пикселей, что соответствует рекомендациям WCAG 2.1 для пользователей с ограниченной моторикой.

Интерфейс упражнений реализует принцип "одна задача — один экран", где все внимание пользователя фокусируется на выполнении текущего задания. Видеопоток с камеры отображается в зеркальном режиме, что упрощает процесс самоконтроля движений. Эталонное изображение жеста сопровождается анимацией плавного появления, что позволяет плавно переключать внимание между демонстрацией и собственными действиями.

Для обеспечения обратной связи используется многоуровневая система уведомлений:

- визуальные маркеры правильного положения пальцев
- цветовая индикация распознанного жеста



- текстовые подсказки о необходимости коррекции движений

### **3.3.2. Взаимодействие с серверной частью**

Архитектура взаимодействия клиентской части с серверными микросервисами построена по принципу "тонкого клиента". Основная бизнес-логика обработки жестов вынесена на клиентскую сторону, что минимизирует объем передаваемых данных. Обмен информацией с сервером осуществляется через REST API с использованием JSON-формата.

Особенностью реализации является гибридный подход к аутентификации:

1. Первичная авторизация через JWT-токены
2. Хранение сессионных данных в HttpOnly cookies
3. Механизм автоматического обновления токенов

Для обработки ошибок сети реализована система экспоненциальной повторной отправки запросов (exponential backoff), что особенно важно для пользователей с нестабильным интернет-соединением. Состояние интерфейса синхронизируется с сервером через механизм оптимистичных обновлений, когда изменения сначала отражаются в интерфейсе, а затем подтверждаются сервером.

## **3.4. Контейнеризация приложения**

### **3.4.1. Контейнеризация сервисов (Docker)**

Система разворачивается в виде набора изолированных контейнеров, каждый из которых отвечает за определенный функциональный модуль [10]. Для сервиса аутентификации используется образ на базе Python 3.9 с предустановленными зависимостями Django и DRF. Особое внимание уделено оптимизации размеров образов:

- многоэтапная сборка для исключения dev-зависимостей;
- использование alpine-образов для минимального размера;

- версионирование тегов для обеспечения воспроизводимости.

Контейнер с сервисом упражнений включает в себя:

- минимальную среду выполнения Python;
- предустановленные бинарные зависимости MediaPipe;
- конфигурационные файлы для работы с SQLite;
- скрипты инициализации базы данных.

### 3.4.2. Настройка docker-compose.yml

Конфигурация оркестрации контейнеров реализована через файл `docker-compose.yml`, который определяет три основных сервиса и общую сетевую инфраструктуру для их взаимодействия. Архитектура развертывания построена с учетом принципов изолированности сервисов и обеспечения надежного взаимодействия между ними.

Структура сервисов:

#### 1. backend (Django):

- собирается из контекста директории `./account-service/backend`;
- выполняет автоматические миграции базы данных при запуске;
- запускает development-сервер на порту 8000;
- монтирует рабочую директорию для hot-reload изменений;
- работает в debug-режиме (`DEBUG=1`);
- подключается к внутренней сети `gesture-network`.

#### 2. frontend (React):

- собирается из контекста `./account-service/frontend`;
- экспонирует порт 3000 для доступа к клиентскому приложению;
- использует `volumes` для синхронизации кода и изоляции `node_modules`;
- настроен для работы с hot-reload через polling (`CHOKIDAR_USEPOLLING`);
- требует настройки `tty` и `stdin_open` для интерактивной работы.

### 3. hand-gesture (Django):

- собирается из контекста `./hand_gesture_project`;
- выполняет миграции и запускает сервер на порту 8001;
- монтирует рабочую директорию для разработки;
- работает в `debug`-режиме;
- подключается к общей сети `gesture-network`.

Сетевая конфигурация реализована через мостовую сеть `bridge`, которая обеспечивает:

- изолированное взаимодействие между контейнерами;
- автоматическое DNS-разрешение имен сервисов;
- возможность масштабирования отдельных компонентов.

Данная конфигурация обеспечивает гибкое развертывание всех компонентов системы одной командой `docker-compose up`, сохраняя при этом возможность индивидуальной настройки каждого сервиса. Для `production`-среды рекомендуется дополнить конфигурацию настройками репликации, логгирования и мониторинга.

# **ЗАКЛЮЧЕНИЕ**

## **Результаты работы**

В ходе выполнения работы над проектом была успешно разработана и внедрена веб-система для восстановления мелкой моторики рук, основанная на микросервисной архитектуре. Было реализовано:

1. создание полнофункционального решения, сочетающего:
  - точное распознавание жестов с использованием MediaPipe Hands;
  - интуитивно понятный интерфейс для пациентов с ограниченными возможностями;
  - безопасную систему аутентификации на основе JWT.
2. разработана оригинальная методика классификации жестов:
  - бинарное кодирование положения пальцев;
  - адаптивный алгоритм подбора упражнений;
  - система визуальной и эмоциональной обратной связи.
3. реализована отказоустойчивая архитектура:
  - изолированные микросервисы в Docker-контейнерах;
  - автоматизированное развертывание через docker-compose;
  - документированное API с поддержкой Swagger UI.

## **Перспективы развития проекта**

Дальнейшее развитие системы предполагает следующие направления совершенствования:

1. расширение функциональности:
  - интеграция с медицинскими CRM-системами;
  - добавление телемедицинских консультаций;
  - разработка мобильной версии приложения.
2. улучшение алгоритмов распознавания:
  - внедрение 3D-анализа движений кисти;

- использование нейросетевых моделей для сложных случаев;
  - добавление коррекции жестов в реальном времени.
3. оптимизация производительности:
- переход на gRPC для межсервисного взаимодействия;
  - кэширование результатов распознавания;
  - поддержка аппаратного ускорения.
4. научно-исследовательские направления:
- адаптация системы для других видов реабилитации;
  - интеграция с VR-технологиями и Smart-часами;
  - разработка персонализированных алгоритмов восстановления.

Разработанная система открывает новые возможности для цифровизации реабилитационного процесса, предлагая доступную альтернативу дорогостоящему оборудованию. Проект имеет значительный потенциал для внедрения в медицинских учреждениях и для домашнего использования, что может существенно улучшить качество реабилитации пациентов с нарушениями мелкой моторики.

Перспективным направлением является коммерциализация проекта через создание SaaS-решения для клиник и реабилитационных центров, а также разработка франшизной модели для тиражирования системы в регионах.

## СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Макарова, М.Р. Проблемы восстановления мелкой моторики у взрослых / М. Р. Макарова // Доктор.Ру. – 2013. – № 10(88). – С. 43-47. – EDN RWIQNX.
2. Ключков, А. С. Современные технологии восстановительной медицины: аппаратные и программные решения / А. С. Ключков, Д. А. Иванов // Вестник восстановительной медицины. – 2022. – № 3(45). – С. 56-64.
3. Петров, А. В. Нейросетевая модель распознавания жестов для реабилитационных приложений / А. В. Петров, Е. К. Смирнова // Труды НИУ ВШЭ. – 2021. – № 8. – С. 112-125. – EDN PLMNBV.
4. Lugaresi C. et al. Mediapipe: A framework for building perception pipelines //arXiv preprint arXiv:1906.08172. – 2019.
5. Кузнецов, А. А. Применение микросервисной архитектуры для построения распределенных информационных систем / А. А. Кузнецов, Д. В. Луцив // Программная инженерия. — 2020. — Т. 11, № 5. — С. 209-217. — DOI: 10.17587/prin.11.209-217. — URL: <https://www.elibrary.ru/item.asp?id=54086509>
6. Fielding R. T. Architectural styles and the design of network-based software architectures. – University of California, Irvine, 2000.
7. Документация Django REST Framework – URL: <https://www.django-rest-framework.org/>.
8. Документация MediaPipe Hands. – URL: <https://mediapipe.readthedocs.io/en/latest/solutions/hands.html>
9. Chen W. et al. A survey on hand pose estimation with wearable sensors and computer-vision-based methods //Sensors. – 2020. – Т. 20. – №. 4. – С. 1074.
10. Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and ulterior software engineering. – 2017. – С. 195-216.