Автономная некоммерческая образовательная организация
высшего образования
«НАУЧНО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «СИРИУС»

Научный центр информационных технологий и искусственного интеллекта
направление «Математическое моделирование в биомедицине и геофизике»

К ЗАЩИТЕ ДОПУСТИТЬ

Научный руководитель направления
«Математическое моделирование в
биомедицине и геофизике»
д.ф.-м.н., профессор, чл.-корр. РАН
_____ Ю.В. Василевский
«_____» _____ 20_____ г.

СОЗДАНИЕ ВЕБ-ИНТЕРФЕЙСА ДЛЯ ПРОВЕДЕНИЯ ЛИНИЙ ПРИШИВАНИЯ СТВОРОК КЛАПАНА АОРТЫ В ОПЕРАЦИИ ОЗАКИ

Магистерская диссертация
по направлению подготовки 01.04.02 Прикладная математика и информатика
(направленность (профиль) «Математическое моделирование в биомедицине и нефтегазовом инжиниринге»)

Консультант по
магистерской диссертации
к.ф.-м.н, доцент БФУ им. И.Канта
_____ Г.В. Копытов
«_____» _____ 20_____ г.

Студент гр. М01ММ-22
_____ А.Н. Дроздов
«_____» _____ 20_____ г.

Научный руководитель
магистерской диссертации
Научный руководитель направления
«Математическое моделирование в
биомедицине и геофизике» Научного
центра информационных технологий и
искусственного интеллекта, д.ф.-м.н.,
_____ Ю.В. Василевский
«_____» _____ 20_____ г.

Федеральная территория «Сириус», 2024

Автономная некоммерческая
образовательная организация высшего образования
«НАУЧНО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «СИРИУС»
Научный центр информационных технологий и искусственного интеллекта
Направление «Математическое моделирование в биомедицине и геофизике»

УТВЕРДИТЬ
Руководитель направления
«Математическое моделирование в биомедицине и нефтегазовом инжиниринге»
д.ф.-м.н., профессор, чл.-корр. РАН
_____ Ю.В. Василевский
«___» _____ 20__ г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
на выполнение выпускной квалификационной работы
обучающегося по направлению подготовки 01.04.02 Прикладная математика и информатика
направленность (профиль) «Математическое моделирование в биомедицине и нефтегазовом инжиниринге»

Дроздова Андрея Николаевича

1. Тема: «Создание веб-интерфейса для проведения линий пришивания створок клапана аорты в операции Озаки»
2. Цель: Разработка пользовательского веб-интерфейса в системе поддержки принятия врачебных решений для операции Озаки с использованием современных фреймворков.
3. Задачи: Изучить специфику операции Озаки и определить функциональные и нефункциональные требования к системе. Проанализировать старый веб-интерфейс, выявить его недостатки и рассмотреть современные подходы к разработке интерфейсов. Описать и реализовать новую кодовую базу с использованием современных фреймворков, описать интеграцию библиотек для решения возникающих задач, обеспечить интеграцию всех компонентов системы.
4. Рабочий график (план) выполнения выпускной квалификационной работы:

№	Перечень заданий	Сроки выполнения
1	Ознакомление с проектом разработки системы поддержки принятия врачебных решений для операции Озаки; анализ проблем старого веб-интерфейса	30.01.2024 - 29.02.2024
2	Разработка функции загрузки и визуализации данных компьютерной томографии органов грудной клетки пациента	01.03.2024 - 30.04.2024

№	Перечень заданий	Сроки выполнения
3	Разработка функции сегментации и визуализации корня аорты пациента	01.05.2024 - 23.05.2024

Дата выдачи: «30» января 2024 г.
Руководитель ВКР:
_____ Ю.В.Василевский

Задание принял к исполнению:
Студент группы М01ММ-22
_____ А.Н. Дроздов
«30» января 2024 г.

Реферат

Выпускная квалификационная работа, 54 страницы, 13 рисунков, 14 источников.

СОЗДАНИЕ ВЕБ-ИНТЕРФЕЙСА ДЛЯ ПРОВЕДЕНИЯ ЛИНИЙ ПРИШИВАНИЯ СТВОРОК КЛАПАНА АОРТЫ В ОПЕРАЦИИ ОЗАКИ

Цель работы:

Разработка пользовательского веб-интерфейса в системе поддержки принятия врачебных решений для операции Озаки с использованием современных фреймворков.

Результаты работы:

В рамках дипломной работы был создан веб-интерфейс, позволяющий загружать и просматривать КТ-данные пациентов в формате DICOM, а также сегментировать и визуализировать аорту. Интерфейс реализован на языке TypeScript с использованием сборщика Vite и фреймворка React. Для работы с 3D-графикой и медицинскими изображениями применяются библиотеки Three.js и Ami.js.

Сокращения, обозначения, термины и определения

API - Application Programming Interface (Интерфейс программирования приложений)

AVD - Aortic Valve Disease (Аортальный порок сердца)

AV-Rec - Aortic Valve Reconstruction (Реконструкция аортального клапана)

CSS - Cascading Style Sheets (Каскадные таблицы стилей)

DICOM - Digital Imaging and Communications in Medicine (Цифровая обработка и передача медицинских изображений)

DOM - Document Object Model (Объектная модель документа)

GCC - Google Closure Compiler (Компилятор Google Closure)

GL - Graphics Library (Графическая библиотека)

HTML - HyperText Markup Language (Язык разметки гипертекста)

HTTPS - HyperText Transfer Protocol Secure (Безопасный протокол передачи гипертекста)

JS - JavaScript (Язык программирования JavaScript)

JSX - JavaScript XML (Расширение синтаксиса JavaScript для использования XML/HTML)

КТ - Компьютерная томография

MVP - Minimum Viable Product (Минимально жизнеспособный продукт)

СППВР - Система поддержки принятия врачебных решений

UI - User Interface (Пользовательский интерфейс)

XML - eXtensible Markup Language (Расширяемый язык разметки)

Шейдеры — это программы, предназначенные для выполнения на графическом процессоре, которые определяют аспекты визуализации 3D-графики, включая цвета, освещение и тени.

2D/3D - Two-dimensional/Three-dimensional (Двумерный/Трехмерный)

Содержание

Введение

Персонализированные модели играют важную роль в биомедицинских задачах, так как они учитывают индивидуальные особенности каждого пациента. Такие модели позволяют формализовать и связать ключевые патофизиологические процессы, оценивать важные для диагностики заболевания параметры и прогнозировать результаты терапевтических или хирургических вмешательств. Однако для успешного использования моделей в клинической практике необходимо обеспечить их валидацию, автоматизацию технологической цепочки от обработки входных данных до получения результата, а также высокую скорость расчетов [1].

Для автоматизации такой технологической цепочки необходимо разработать систему поддержки принятия врачебных решений (СППВР). Задача разработки СППВР включает в себя не только разработку математической модели, но и создание удобного интерфейса для взаимодействия с врачами, что особенно важно для эффективного применения в клинической практике [2].

Для успешного внедрения СППВР в клиническую практику необходимо, чтобы система удовлетворяла следующим техническим критериям: масштабируемость, развертываемость, поддерживаемость, сопровождаемость, совместимость внутренних модулей. Эти критерии распространяются и на внутренние модули системы, в том числе на интерфейс.

Помимо технических критериев, самым важным является практическая значимость для врачей: разрабатываемая система должна уметь эффективно решать актуальную проблему врачей.

В рамках данной работы было предложено проанализировать проблемы старого веб-интерфейса СППВР для операции Озаки, изучить современные подходы к созданию веб-интерфейса и реализовать новую кодовую базу на основе этих фреймворков или доработать существующую.

Цель данной работы — разработка пользовательского веб-интерфейса в системе поддержки принятия врачебных решений для операции Озаки с использованием современных фреймворков.

Список задач для достижения цели:

1. Анализ предметной области: Изучить специфику операции Озаки и определить функциональные и нефункциональные требования к системе;
2. Анализ проблем предыдущего решения и современных подходов к созданию клиентских веб-приложений: Оценить старый веб-интерфейс, выявить его недостатки и рассмотреть современные подходы к разработке интерфейсов;
3. Реализация веб-интерфейса: Описать и реализовать новую кодовую базу с использованием современных фреймворков, описать интеграцию библиотек для решения возникающих задач, обеспечить интеграцию всех компонентов системы.

1 ОПЕРАЦИЯ ОЗАКИ. СППВР ДЛЯ ОПЕРАЦИИ ОЗАКИ И РАЗРАБОТКА ТРЕБОВАНИЙ К ВЕБ-ИНТЕРФЕЙСУ ДЛЯ СППВР

В первой главе рассматривается метод Озаки, его особенности и преимущества. Обсуждаются возможности применения персонализированной модели для операции Озаки, которая может быть внедрена в систему поддержки принятия врачебных решений (СППВР). Делается акцент на том, что основное внимание данной работы будет уделено разработке веб-интерфейса для данной СППВР. Также в этой главе описываются основные требования к создаваемому веб-интерфейсу, которые обеспечат его функциональность и удобство использования для медицинских специалистов.

1.1 Операция Озаки

Аортальный порок сердца (AVD - Aortic Valve Disease) составляет почти половину всех заболеваний клапанов сердца, затрагивая миллионы людей по всему миру. На протяжении многих лет замена аортального клапана (AVR - Aortic Valve Replacement) была признанным стандартом лечения тяжелых случаев AVD, включающим замену нативного аортального клапана на биологический или механический протез. С каждым годом количество процедур по лечению AVD растет, и по прогнозам, к 2050 году их число достигнет 850 000 [3; 4].

Аортальный порок сердца бывает двух типов: аортальный стеноз и аортальная регургитация (Рисунок 1.1 [5]) [6].

В случае аортального стеноза стенки клапана утолщаются и становятся менее гибкими, что приводит к сужению прохода для крови, вытекающей из левого желудочка сердца в аорту. Это препятствует нормальному кровотоку и заставляет сердце работать с повышенной нагрузкой для обеспечения необходимого кровообращения. Симптомы аортального стеноза могут включать одышку, боли в груди (стенокардию), обмороки и сердечную недостаточность. Причинами аортального стеноза могут быть врожденные пороки сердца, возрастные изменения клапана, ревматические заболевания и кальцификация клапана [6].

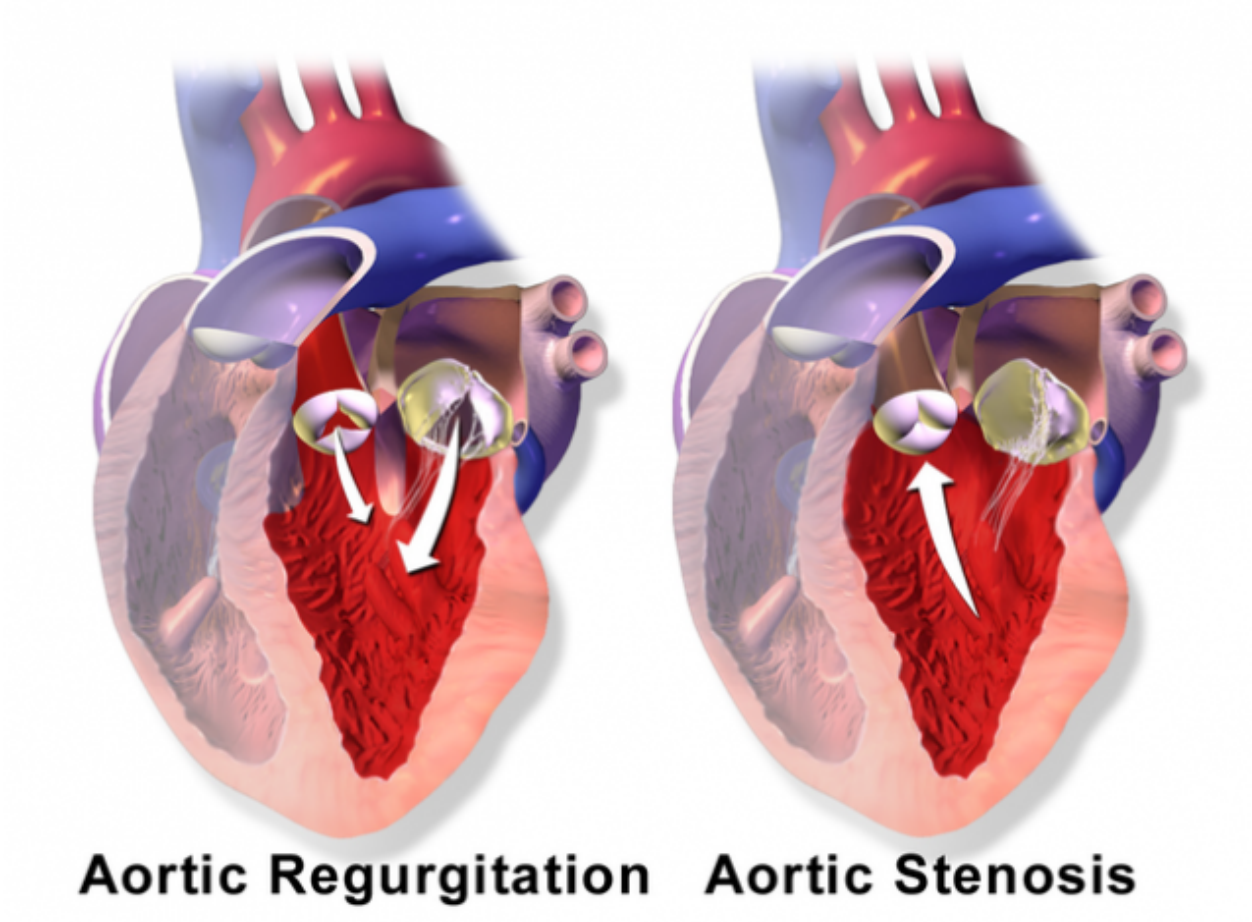

Рисунок 1.1. Типы аортального порока сердца

Аортальная регургитация возникает, когда клапан не закрывается полностью, что приводит к обратному току крови из аорты в левый желудочек во время диастолы (фазы расслабления сердца). Это вызывает перегрузку объемом в левом желудочке и может привести к его увеличению и ослаблению. Симптомы аортальной регургитации могут включать усталость, одышку, учащенное сердцебиение и отеки. Причины аортальной регургитации могут включать инфекционный эндокардит, ревматические заболевания, травмы клапана и дегенеративные изменения клапана [6].

Операция Озаки, также известная как реконструкция аортального клапана (AV-Rec - Aortic Valve Reconstruction), представляет собой инновационную хирургическую технику, которая реконструирует аортальный клапан с использованием собственного перикарда пациента, чтобы преодолеть ограничения существующих протезов. Впервые предложенная доктором С. Озаки, эта процедура включает замену створок аортального клапана на три створки, вырезанные из аутоперикарда пациента. Опубликованные результаты операции Озаки весьма обнадеживают. Эта процедура подходит для лю-

бого типа дефекта и этиологии аортального клапана (ревматической, дегенеративной, эндокардита и других), независимо от того, является ли клапан двустворчатым или трехстворчатым.

Методика, используемая при проведении операции Озаки, включает вырезание части перикарда пациента необходимого размера, который затем химически обрабатывается, из него выкраиваются и вшиваются новые створки на место аортального клапана. Текущие шаблоны для вырезания створок, представленные на рисунке 1.2 [7] и предложенные доктором С. Озаки, предполагают значительно больший размер, чем нативный клапан. Это обеспечивает большую зону коаптации (смыкания) клапанов, но может привести к избыточному размеру створок, что мешает нормальному функционированию клапана [3; 4].

Рисунок 1.2. Шаблоны створок

Одним из основных преимуществ операции Озаки являются превосходные гемодинамические показатели сразу после операции и отсутствие необходимости в послеоперационной антикоагуляции. Средний возраст пациентов, подвергшихся данной процедуре, составляет 67.7 лет. Этиология заболевания включает аортальный стеноз в 61.7% случаев, аортальную недостаточность в 31.1%, и сочетание этих состояний в 7.2%. В кардиологическом

центре Monzino с октября 2014 года по февраль 2020 года операция Озаки была выполнена 71 пациенту. Ни одного случая госпитальной смерти не было зафиксировано. Отсутствие серьёзных осложнений, связанных с клапаном, наблюдалось в 97% случаев. На контрольной эхокардиографии через три месяца после операции показатели градиентов давления на аортальном клапане и трансвальвулярной скорости были значительно ниже, чем на эхокардиографии перед выпиской (10.93 ± 5.38 мм рт. ст. против 16.24 ± 7.67 мм рт. ст., соответственно) [8].

Исследования показывают, что среднесрочные результаты после операции Озаки оптимальны с точки зрения смертности, градиентов давления на аортальном клапане, отсутствия серьезных неблагоприятных событий, связанных с клапаном, и рецидива аортальной недостаточности [8].

Одним из ключевых преимуществ является то, что операция Озаки может быть выполнена без необходимости применения антикоагулянтов после операции, что особенно важно для пациентов, которые имеют противопоказания к такому лечению. Тем не менее, существуют и недостатки. Например, требуется период обучения для хирургов, так как первые 20 пациентов требуют тщательного наблюдения и контроля на начальных этапах [9].

1.2 Система поддержки принятия врачебных решений для операции Озаки на основе персонализированной модели

Створки клапана вырезаются вручную в реальном времени, а подборка их формы происходит также вручную. Это означает, что пациент находится на системе искусственного кровообращения в течение некоторого времени.

Оптимизация формы шаблона створок на основе КТ-изображений сердца пациента до операции позволит сократить время операции и обеспечить оптимальную для пациента зону коаптации. Это подчеркивает необходимость разработки персонализированной модели закрытия аортального клапана, которая учитывает индивидуальные особенности анатомии корня аорты пациента и может проводить расчеты в оперативном режиме. Таким образом, создание системы поддержки принятия врачебных решений на основе персонализированных моделей способно значительно улучшить результаты операции Озаки и снизить риски, связанные с избыточным размером створок [1; 10].

Одним из ключевых решений для внедрения персонализированной модели при проведении операции Озаки является разработка системы поддержки принятия врачебных решений (СППВР). Такая система представляет собой конечный продукт, предназначенный для использования в клинической практике, и существенно облегчает процесс подготовки и проведения операции, что может увеличить эффективность хирургического вмешательства, а также сократить время выполнения операции, за счет чего снизить вероятность возникновения осложнений [11].

Общая схема СППВР представлена на рисунке 1.3. Из схемы видно, что система включает в себя множество технических задач, что позволяет разделить её на несколько модулей.

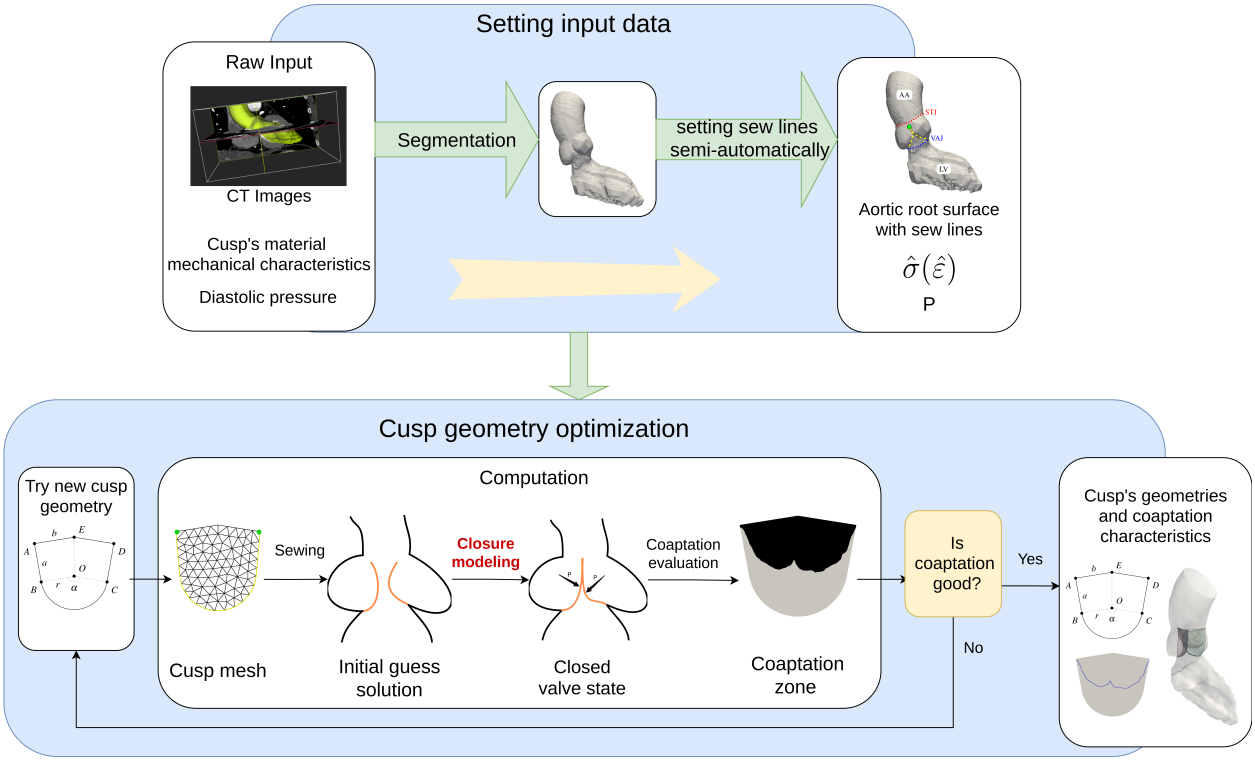

Рисунок 1.3. Схема СППВР для операции Озаки

Задача была разбита на следующие модули:

1. Модуль сегментации: сегментации корня аорты из КТ данных пациента;
2. Модуль пользовательского интерфейса (UI): Предоставляет интерфейс для взаимодействия врача с системой;
3. Модуль расчета геометрии: Модуль вычисления для определения оптимальных размеров и формы створок клапана на основе входных данных.

Проект разработки системы поддержки принятия врачебных решений разделен на несколько частей, над каждой из которых работают различные специалисты. В рамках данной работы внимание уделяется созданию веб-интерфейса, который обеспечивает взаимодействие врача с системой и предоставляет все необходимые инструменты для подготовки к операции.

Выбор веб-технологий для разработки пользовательского интерфейса имеет множество преимуществ:

- Веб-приложения работают на любой операционной системе, будь то Windows, macOS или Linux. Это достигается благодаря использованию стандартных веб-технологий, таких как HTML, CSS и JavaScript;
- Обновление веб-приложений происходит централизованно на сервере, что исключает необходимость обновления на каждом отдельном устройстве. Это значительно упрощает процесс внедрения новых функций и исправлений;
- Веб-приложения легко интегрируются с другими модулями системы благодаря клиент-серверной архитектуре. Это позволяет обеспечивать взаимодействие между различными компонентами системы, такими как модуль сегментации и модуль расчета геометрии;
- Веб-приложения могут использовать современные методы защиты данных, включая шифрование и многофакторную аутентификацию. Это особенно важно при работе с чувствительными медицинскими данными.

1.3 Разработка требований

Так как веб-интерфейс напрямую взаимодействует с врачом, необходимо определить и задокументировать основные функции, которые требуются от системы, чтобы обеспечить её эффективное использование в клинической практике. Для этого используется диаграмма вариантов использования, которая иллюстрирует взаимодействие пользователя (врача) с системой.

Основные функции, которые были выделены:

- Загрузка КТ данных пациента. Эта функция позволяет загружать компьютерно-томографические (КТ) данные пациента в систему в

формате DICOM. Данная функция включает в себя сохранение загруженных файлов для переиспользования в других сессиях работы с приложением;

- Сегментация аорты. Функция сегментации аорты позволяет выделить аорту на основе загруженных КТ данных.Требования: интерактивная 3D визуализация сегментированной аорты, сохранение результатов сегментации для последующего использования, возможность настройки параметров сегментации для получения оптимальных результатов;
- Проведение линий пришивания. Эта функция позволяет пользователю проводить линии пришивания на сегментированных данных аорты. Эти линии определяют оптимальные места крепления створок клапана. Требования: интуитивно понятные инструменты для проведения и корректировки линий пришивания, возможность удаления и изменения линий, сохранение всех изменений и результатов для последующего анализа;
- Моделирование движения створок. Функция моделирования движения створок позволяет визуализировать и анализировать движение створок аортального клапана на основе проведённых линий пришивания и сегментированных данных. Это помогает оценить функциональность и эффективность предложенной модели до проведения операции. Требования: визуализация результатов моделирования в реальном времени, сохранение модели для дальнейшего анализа и использования, интерактивные инструменты для настройки параметров моделирования.

Основные функциональные требования представлены на диаграмме прецедентов на рисунке 1.4.

При разработке системы необходимо учитывать не только функциональные, но и нефункциональные требования, которые обеспечивают качество и удобство использования приложения. Важно выделить следующие нефункциональные требования, учитывая специфику работы с 3D графикой и медицинскими данными.

Одним из важнейших аспектов является поддержание оптимальной производительности приложения, так как работа с 3D графикой требует зна-

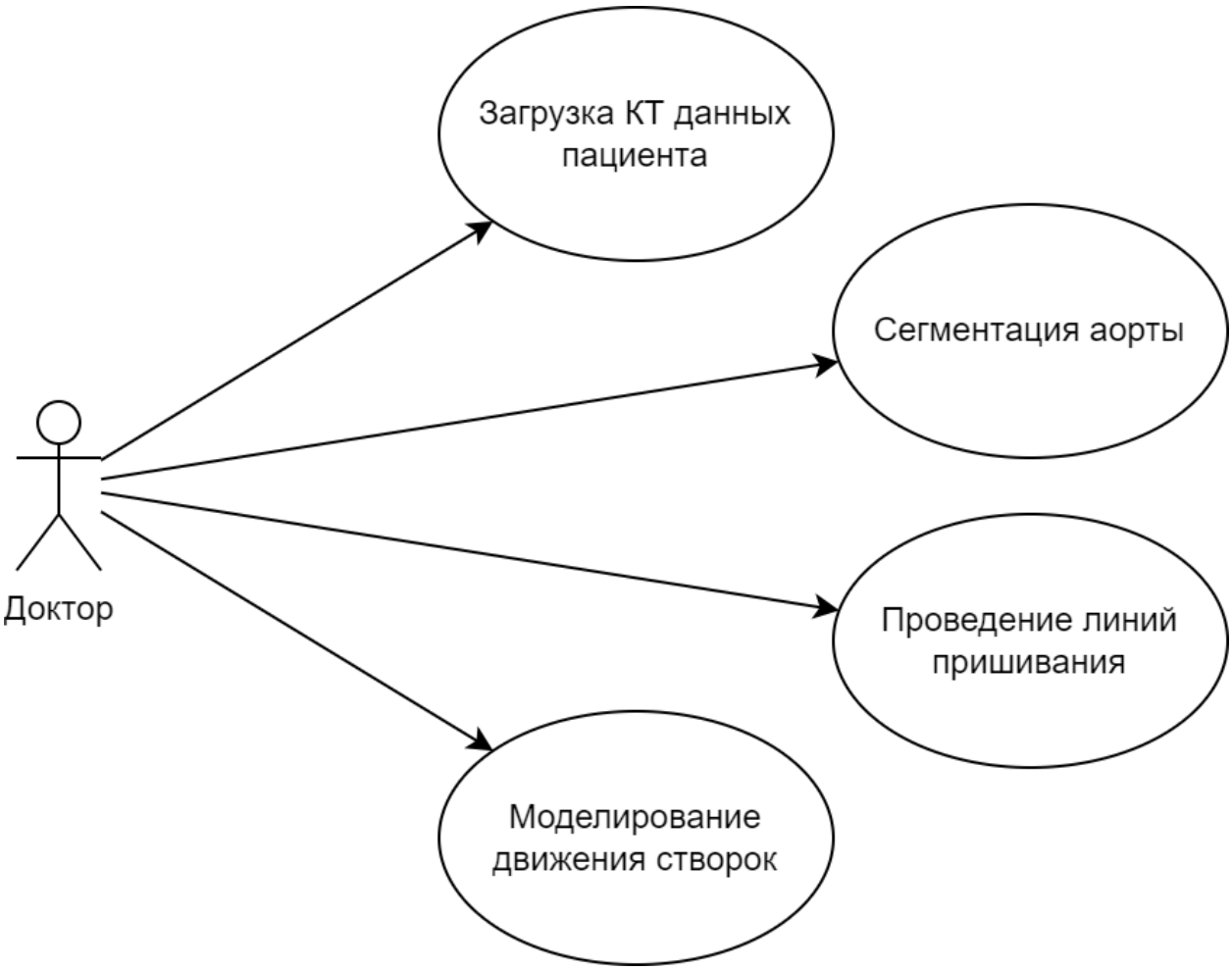

Рисунок 1.4. Диаграмма прецедентов пользовательского интерфейса

чительных вычислительных ресурсов. Количество кадров в секунду (FPS) должно быть стабильным и не падать до критических значений, при которых работа с приложением становится невозможной. Для обеспечения этого требования необходимо, чтобы в программе были:

- Оптимизация рендеринга 3D сцен;
- Эффективное использование ресурсов процессора и видеокарты.

Пользовательский интерфейс должен быть интуитивно понятным и удобным для использования. Важно, чтобы все функции и инструменты были легко доступны и понятны даже для пользователей с минимальным опытом работы с подобными системами. Это включает:

- Логичное расположение элементов управления;
- Ясные и понятные метки и подсказки;
- Последовательность и предсказуемость поведения интерфейса.

Время отклика приложения на пользовательские действия должно быть минимальным для коротких операций, чтобы обеспечивать плавный и непрерывный опыт работы.

Для длительных операций, таких как сегментация аорты или моделирование движения створок, необходимо обеспечить информирование пользователя о ходе выполнения. Это может быть реализовано через прогресс-бары или индикаторы выполнения.

Приложение должно быть надежным и устойчивым к ошибкам. При возникновении ошибок должен быть удобный инструмент для определения версии приложения и модулей, для формирования баг-репортов.

2 СОВРЕМЕННЫЕ ТЕХНОЛОГИИ И ПОДХОДЫ В РАЗРАБОТКЕ ВЕБ-ИНТЕРФЕЙСОВ

Во второй главе рассматриваются основы современной фронтенд-разработки, включая использование HTML, CSS, JavaScript (JS) и TypeScript (TS). Описаны основные инструменты и подходы, такие как сборщики и фреймворки, применяемые в разработке веб-интерфейсов. Проведен анализ старой кодовой базы и выявлены ее основные проблемы. Также выполнен обзор современных фреймворков, после чего выбран наиболее подходящий фреймворк и сборщик для создания нового веб-интерфейса, учитывая требования и особенности проекта.

2.1 Основы создания приложений в браузере

2.1.1 HTML, Css, JavaScript

Создание веб-приложений для браузеров основывается на использовании трех основных технологий: HTML, CSS и JavaScript. Эти технологии являются фундаментом для разработки интерактивных и динамических веб-страниц.

HTML — это язык разметки гипертекста, который используется для создания структуры веб-страниц. С помощью HTML можно определять различные элементы DOM-дерева (Document Object Model), такие как заголовки, абзацы, изображения, ссылки, формы и многое другое. HTML задает базовую структуру страницы, на которую затем накладываются стили и поведение.

CSS — это каскадные таблицы стилей, которые используются для определения внешнего вида веб-страниц. С помощью CSS можно задавать стили для HTML-элементов, такие как цвет, шрифт, размеры, отступы, выравнивание и т.д. CSS позволяет отделить содержание страницы от ее представления, что делает разработку и поддержку веб-приложений более удобной и эффективной.

JavaScript — это язык программирования, который поддерживается всеми современными браузерами и позволяет создавать интерактивные и динамические веб-страницы. С помощью JavaScript можно манипулировать

DOM-деревом, добавлять анимации, обрабатывать события, взаимодействовать с сервером и многое другое. JavaScript делает веб-страницы живыми, позволяя пользователям взаимодействовать с ними в реальном времени.

Можно сказать, что средств HTML, CSS и JavaScript достаточно для создания интерактивного веб-сайта. Однако в современных реалиях часто используются фреймворки для создания клиентских веб-приложений.

Фреймворки предназначены для упрощения процесса разработки приложений, сайтов и сервисов. Вместо создания модулей приложения с нуля, разработчики могут использовать готовые шаблоны фреймворков, которые формируют удобную рабочую среду. Эти фреймворки предоставляют высокоуровневые абстракции и инструменты, которые позволяют более эффективно управлять состоянием приложения, взаимодействовать с DOM-деревом и реализовывать реактивность данных.

Подробный обзор существующих фреймворков описан в разделе 2.3.

2.1.2 Сборщики

Сборщики (bundlers) JavaScript играют важную роль в современных веб-разработках, предоставляя разработчикам инструменты для объединения, оптимизации и управления ресурсами проекта. Сборщики позволяют обрабатывать различные типы файлов, такие как JavaScript, CSS, HTML и изображения, а затем объединять их в оптимизированные пакеты для развертывания на сервере.

Основные функции сборщиков:

- Объединение файлов: Сборщики позволяют объединять множество файлов в один или несколько оптимизированных пакетов. Это снижает количество HTTP-запросов, необходимых для загрузки страницы, что улучшает производительность;
- Минификация: Минификация уменьшает размер файлов, удаляя пробелы, комментарии и другие незначительные элементы кода. Это сокращает время загрузки и объем передаваемых данных;
- Транспиляция: Сборщики могут транспилировать код из современных стандартов JavaScript (ES6+) в более старые версии, поддержи-

ваемые всеми браузерами. Это обеспечивает совместимость кода с широким спектром устройств;

- Обработка CSS и HTML: Сборщики могут обрабатывать CSS и HTML файлы, применяя такие техники, как автопрефиксация, минификация и объединение стилей, что упрощает управление стилями и улучшает производительность;
- Поддержка модулей: Сборщики поддерживают модульную архитектуру, что позволяет разбивать код на независимые модули и управлять зависимостями между ними;
- Hot Module Replacement (HMR): HMR позволяет обновлять модули в реальном времени без полной перезагрузки страницы, что ускоряет процесс разработки и тестирования.

Популярные сборщики:

- Webpack — один из самых популярных и мощных сборщиков JavaScript. Он поддерживает все основные функции, такие как объединение файлов, минификация, транспиляция и HMR. Webpack имеет гибкую конфигурацию и позволяет настроить сборку под любые потребности проекта;
- Vite — это современный сборщик и инструмент для разработки, созданный для обеспечения высокой скорости и эффективности. Vite разработан с учетом потребностей современных веб-приложений и использует преимущества ES-модулей, поддерживаемых браузерами. Vite из коробки поддерживает работу с TypeScript, JSX, CSS и многими другими популярными технологиями, что упрощает настройку и интеграцию.

Сборщики JavaScript являются неотъемлемой частью современного процесса веб-разработки. Они позволяют оптимизировать код, улучшить производительность и упростить управление проектом. Webpack, Vite — это лишь некоторые из популярных инструментов, каждый из которых имеет свои особенности и преимущества. Выбор сборщика зависит от конкретных потребностей проекта и предпочтений разработчиков.

В итоге, для разработки веб-интерфейса был выбран сборщик Vite из-за плюсов, описанных выше.

2.1.3 Typescript

TypeScript — это язык программирования, который расширяет JavaScript, добавляя статическую типизацию и другие возможности. Разработанный Microsoft и впервые выпущенный в 2012 году, TypeScript быстро завоевал популярность среди разработчиков благодаря улучшенной безопасности кода, автодополнению и интеграции с современными инструментами разработки.

Основные особенности языка заключатся в:

- Статическая типизация: TypeScript позволяет разработчикам задавать типы для переменных, функций, параметров и возвращаемых значений. Это помогает обнаруживать ошибки на этапе компиляции, что снижает количество багов в коде и упрощает его поддержку;
- Поддержка современных стандартов JavaScript: TypeScript поддерживает все последние версии JavaScript, включая ES6 и выше, и компилируется в чистый JavaScript, который может быть выполнен в любом браузере;
- Интерфейсы и типы: TypeScript вводит понятия интерфейсов и пользовательских типов, что позволяет создавать более структурированный и хорошо документированный код. Интерфейсы помогают определить контракт для объектов, который должен быть соблюден, что обеспечивает дополнительную проверку типов;
- Классы и модули: TypeScript расширяет возможности работы с классами и модулями, добавляя поддержку наследования, модификаторов доступа (public, private, protected) и других концепций объектно-ориентированного программирования;
- Генерики: TypeScript поддерживает обобщенные типы, которые позволяют создавать компоненты, работающие с различными типами данных, обеспечивая при этом строгую типизацию;
- Аннотации типов: Разработчики могут добавлять аннотации типов, чтобы указать ожидаемый тип данных для переменных, параметров и возвращаемых значений функций. Это помогает избежать ошибок и делает код более читаемым;

- Интеграция с редакторами кода: TypeScript интегрируется с популярными редакторами кода, такими как Visual Studio Code, обеспечивая автодополнение, рефакторинг и подсказки по типам в реальном времени.

Преимущества использования TypeScript:

- Улучшенная надежность кода: Благодаря статической типизации и проверке типов на этапе компиляции, TypeScript помогает обнаруживать и устранять ошибки до выполнения кода, что повышает надежность приложений;
- Удобство разработки: TypeScript улучшает автодополнение и навигацию по коду в редакторах, что ускоряет процесс разработки и уменьшает количество ошибок;
- Повышенная читаемость и поддерживаемость: Аннотации типов и интерфейсы делают код более самодокументируемым и понятным, что облегчает его поддержку и развитие;
- Совместимость с JavaScript: TypeScript является строгим супермножеством JavaScript, поэтому любой JavaScript-код является валидным TypeScript-кодом. Это упрощает миграцию существующих проектов на TypeScript;
- Активное сообщество и поддержка: TypeScript имеет большое и активное сообщество разработчиков, а также официальную поддержку от Microsoft, что обеспечивает доступ к многочисленным ресурсам, библиотекам и инструментам.

Из минусов языка можно отметить дополнительную сложность изучения типизации, а также потребность в написании большего количества кода, по сравнению с JavaScript.

Однако в современных реалиях использование TypeScript является почти стандартом, так как в долгосрочной перспективе проект на TypeScript намного легче поддерживать.

2.2 Анализ проблем предыдущей кодовой базы

На момент начала разработки существовала кодовая база, представляющая собой веб-интерфейс для взаимодействия с пользовате-

лем.Функциональность приложения была готова на 90% от запланированной минимально жизнеспособной версии (MVP). Приложение позволяет входить в личный кабинет доктора, загружать DICOM пациента, сегментировать аорту с выбором порога сегментации и проводить линию пришивания в ручном режиме (рисунок 2.1). После этого координаты опорных точек линии должны передаваться для моделирования и последующей визуализации. Однако, внедрение новых функций становится проблемой.

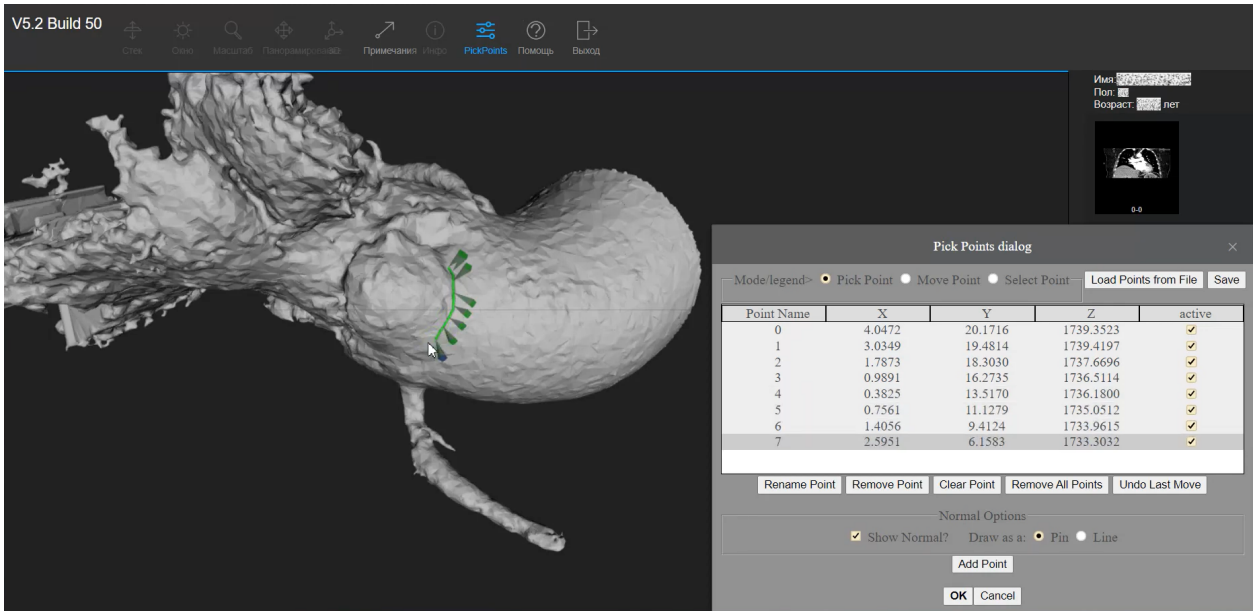

Рисунок 2.1. Установка линий пришивания

Основные проблемы заключаются в следующем:

- Устаревшие технологии: Приложение использует Google Closure Compiler (GCC) и UI-библиотеки Google, которые ограничивают модульность кода и усложняют его поддержку. Эти технологии не предоставляют современных возможностей сборки, таких как hot module replacement;
- Отсутствие модульности: Кодовая база имеет низкую модульность, что повышает трудоемкость разработки и усложняет добавление новых функциональностей;
- Сложность интеграции: Ограниченная возможность интеграции с современными инструментами и библиотеками, что увеличивает время и усилия, необходимые для поддержания и расширения функциональности приложения.

В связи с этими проблемами встал вопрос: продолжать дорабатывать текущий проект до полного завершения MVP, внедряя в него современные сборщики и поддерживая его в дальнейшем, либо переписать код на новые технологии.

После обсуждения и поиска решений стало очевидно, что объединение Google Closure Compiler с современными сборщиками и фреймворками — задача сложная и трудоемкая. В рамках дипломной работы основной целью было изучение современных подходов и использование их на практике. Поэтому было принято решение переписать старый функционал на новом современном фреймворке.

Для успешного перехода необходимо было выбрать фреймворк, который покрывает следующие требования:

- Модульность: Возможность разделения кода на независимые модули, что облегчает его поддержку и масштабирование;
- Современные инструменты сборки: Поддержка инструментов, таких как Vite, Webpack или другие, которые обеспечивают быструю сборку и разработку с поддержкой hot module replacement;
- Реактивность: Поддержка реактивного программирования, что позволяет легко управлять состоянием приложения и обновлением интерфейса.
- Типизация: Возможность использования TypeScript для повышения предсказуемости кода и улучшения отладки;
- Широкая поддержка сообщества: Наличие активного сообщества разработчиков и обширной документации для упрощения процесса обучения и решения возникающих проблем.

Переход на новый фреймворк позволил бы улучшить качество кода, повысить его масштабируемость и облегчить поддержку, а также ускорить разработку новых функциональностей. Это обеспечит более стабильную и предсказуемую основу для дальнейшего развития приложения и других проектов.

2.3 Современные фреймворки для разработки клиентских веб-приложений

В этом разделе рассматриваются современные клиентские веб-фреймворки, которые активно используются для разработки веб-приложений. Основной особенностью этих фреймворков является их способность эффективно управлять реактивностью данных, что значительно упрощает создание и управление пользовательскими интерфейсами.

Реактивность данных является одной из ключевых характеристик современных клиентских веб-фреймворков. Эта особенность позволяет разработчикам легко и эффективно манипулировать DOM-деревом, что в свою очередь упрощает создание динамических и интерактивных пользовательских интерфейсов. Основные преимущества реактивности данных включают[12; 13]:

- Автоматическое обновление интерфейса: Когда данные изменяются, соответствующие элементы пользовательского интерфейса автоматически обновляются без необходимости ручного вмешательства;
- Улучшенная производительность: Фреймворки оптимизируют процесс обновления DOM, что позволяет значительно повысить производительность приложений;
- Реактивное программирование упрощает создание сложных пользовательских интерфейсов, делая код более читаемым и поддерживаемым;
- Снижение количества ошибок: Благодаря автоматическому обновлению интерфейса уменьшается вероятность возникновения ошибок, связанных с некорректным состоянием данных.

Среди множества доступных фреймворков для разработки клиентских веб-приложений выделяются три основных: React, Angular и Vue. Каждый из этих фреймворков имеет свои особенности и преимущества, которые делают их популярными среди разработчиков.

2.3.1 React

React — это открытая и бесплатная JavaScript библиотека, разработанная в 2013 году. React позволяет разработчикам создавать компоненты, которые могут использоваться повторно в различных частях приложения. Основная идея React заключается в облегчении процесса разработки динамических и интерактивных интерфейсов за счёт использования компонентного подхода и виртуального DOM.

Из ключевых особенностей можно выделить следующие: JSX (JavaScript XML), компоненты и Виртуальный DOM.

JSX является расширением JavaScript и позволяет разработчикам писать HTML-подобный код в файлах JavaScript. Это упрощает создание и визуализацию интерфейсов, так как JSX позволяет комбинировать логические и визуальные части компонентов в одном файле.

В React все интерфейсы построены на компонентах. Компоненты могут содержать состояние (state) и свойства (props). Состояние компонента управляется внутри компонента, а свойства передаются от родительского компонента к дочернему в виде неизменяемых данных. Компоненты могут быть функциональными или классовыми. В последних версиях React предпочтительнее использовать функциональные компоненты и хуки (hooks) для управления состоянием и поведением компонентов.

React компоненты проходят через несколько стадий жизненного цикла, что позволяет разработчикам выполнять код на различных этапах. Основные методы жизненного цикла включают:

- ComponentDidMount(): вызывается после монтирования компонента;
- ShouldComponentUpdate(): позволяет определить, должен ли компонент обновляться;
- ComponentDidUpdate(): вызывается после обновления компонента;
- ComponentWillUnmount(): вызывается перед размонтированием и удалением компонента.

Виртуальный DOM представляет собой легковесную копию реального DOM. Это позволяет React эффективно обновлять и рендерить компоненты, минимизируя количество изменений в реальном DOM и, следовательно, улучшая производительность приложения.

React основывается на компонентной архитектуре, что позволяет разработчикам разбивать пользовательский интерфейс на независимые, повторно используемые части. Это облегчает поддержку и масштабирование приложений. Компоненты могут быть встроены друг в друга, создавая иерархию компонентов, что упрощает организацию кода и повторное использование компонентов.

React сам по себе не является полноценным фреймворком, но поддерживает интеграцию с множеством сторонних библиотек и инструментов, что обеспечивает значительную гибкость при построении приложений.

Из основных преимуществ можно выделить:

- Высокая производительность за счет виртуального DOM;
- Гибкость и модульность благодаря компонентному подходу;
- Большая и активная сообщество разработчиков, множество готовых решений и библиотек.

Из недостатков можно выделить

- Крутая кривая обучения для новичков, особенно для тех, кто не знаком с JavaScript;
- Быстрое изменение экосистемы и необходимость постоянного обновления знаний.

Отметим, что ограниченная встроенная функциональность требует использования дополнительных библиотек для реализации многих возможностей. Данный пункт можно отнести как к преимуществам, так и к недостаткам, так как это обеспечивает большую гибкость и возможность адаптировать приложение под конкретные задачи, но может затруднить выбор оптимального решения, потому что разработчикам приходится тратить время на исследование и тестирование различных вариантов.

В заключение, React является мощным инструментом для разработки современных веб-приложений благодаря своей гибкости, производительности и активному сообществу. Он позволяет эффективно создавать сложные и интерактивные пользовательские интерфейсы, что делает его одним из самых популярных выборов среди разработчиков.

2.3.2 Angular

Angular — это мощный фреймворк для разработки одностраничных веб-приложений, разработанный и поддерживаемый Google. Он основан на языке TypeScript и позволяет создавать масштабируемые и поддерживаемые приложения. Angular отличается строгой архитектурой и предоставляет разработчикам полный набор инструментов для создания сложных веб-приложений.

Из ключевых особенностей можно выделить следующие: встроенная поддержка Typescript, модульная архитектура и декларативные шаблоны и двустороннее связывание данных.

Angular использует модульную архитектуру, что позволяет разделять приложение на независимые и переиспользуемые модули. Это облегчает поддержку и масштабирование приложений. Главный модуль, как правило, называется AppModule, и он может импортировать другие модули, необходимые для работы приложения.

Angular использует декларативные шаблоны, основанные на расширениях HTML. Эти шаблоны позволяют разработчикам связывать данные с DOM элементами с помощью Angular директив, таких как *ngIf, *ngFor, [(ngModel)] и других. Это упрощает создание интерактивных и динамических пользовательских интерфейсов.

Одной из ключевых особенностей Angular является двустороннее связывание данных. Это означает, что изменения в модели автоматически отражаются в представлении и наоборот. Это обеспечивает синхронизацию данных между моделью и пользовательским интерфейсом, что делает разработку более интуитивной и уменьшает количество ручного кода.

Angular приложения состоят из компонентов и директив. Компоненты содержат логику и представление, объединенные в одном классе, и могут быть вложены друг в друга для создания сложных интерфейсов. Директивы позволяют изменять поведение и внешний вид DOM элементов, добавляя дополнительную функциональность к стандартным HTML элементам.

Angular компоненты проходят через несколько стадий жизненного цикла, что позволяет разработчикам выполнять код на различных этапах. Основные методы жизненного цикла включают:

- NgOnInit(): вызывается после инициализации компонента;
- NgOnChanges(): вызывается при изменении входных данных;
- NgDoCheck(): позволяет обнаруживать и реагировать на изменения, которые Angular не отслеживает автоматически;
- NgAfterContentInit(): вызывается после вставки содержимого в компонент;
- NgAfterViewInit(): вызывается после инициализации представления компонента;
- NgOnDestroy(): вызывается перед уничтожением компонента.

Angular активно использует концепцию сервисов для отделения бизнес-логики от логики представления. Сервисы можно внедрять в компоненты с помощью механизма Dependency Injection (DI), что обеспечивает переиспользование кода и улучшает тестируемость приложений.

Приложения организуются в модули, что позволяет разбивать функциональность на логические части. Модульная система облегчает управление зависимостями и улучшает производительность за счёт отложенной загрузки модулей. Angular Router позволяет создавать маршруты для навигации между различными частями приложения, поддерживая как одностраничные приложения (SPA), так и многостраничные приложения (MPA).

Angular поддерживает интеграцию с множеством сторонних библиотек и инструментов. Для управления состоянием можно использовать NgRx, который предоставляет архитектуру, основанную на паттерне Redux. Angular Material предоставляет коллекцию UI компонентов, основанных на принципах Material Design, что упрощает создание стильных и удобных интерфейсов.

Из основных преимуществ можно выделить:

- Строгая архитектура и модульная система, облегчающие поддержку и масштабирование приложений;
- Декларативные шаблоны и двустороннее связывание данных, упрощающие разработку пользовательских интерфейсов;
- Богатый набор встроенных инструментов и библиотек, таких как Angular CLI и Angular Material.

Из недостатков можно выделить:

- Крутая кривая обучения, особенно для разработчиков, не знакомых с TypeScript и концепцией DI;
- Большой размер и сложность, что может увеличивать время загрузки и ресурсоёмкость приложений.

В заключение, Angular является мощным фреймворком для создания современных веб-приложений благодаря своей архитектуре, функциональности и поддержке со стороны Google. Он позволяет создавать масштабируемые и поддерживаемые приложения, что делает его популярным выбором среди крупных компаний и организаций.

2.3.3 Vue

Vue — это прогрессивный JavaScript-фреймворк, созданный в 2014 году, предназначенный для создания пользовательских интерфейсов. Vue обладает гибкостью и простотой интеграции, что делает его популярным среди разработчиков, особенно для создания одностраничных приложений (SPA). Vue сочетает в себе лучшие качества других фреймворков, таких как Angular и React, предлагая при этом свою уникальную архитектуру и подходы.

Основные особенности — это реактивные данные, шаблоны и компоненты.

Vue использует реактивную систему данных, что позволяет автоматически обновлять представление при изменении состояния модели. Это делает разработку более интуитивной и сокращает количество ручного кода для синхронизации данных и представления.

Vue, как и Angular, использует декларативные шаблоны, основанные на расширениях HTML. Эти шаблоны позволяют связывать данные с DOM элементами с помощью директив, таких как v-bind, v-model, v-if, v-for, и других. Это упрощает создание динамических и интерактивных интерфейсов.

Компоненты являются основными строительными блоками Vue-приложений. Каждый компонент инкапсулирует свою логику, представление и стили, что облегчает их переиспользование и поддержку. Компоненты могут быть вложены друг в друга, создавая сложные пользовательские интерфейсы из простых и независимых частей.

Vue использует односторонний поток данных, что упрощает понимание и отладку приложений. Данные всегда передаются сверху вниз от родительских компонентов к дочерним через свойства (props). Это делает код более предсказуемым и снижает вероятность ошибок при управлении состоянием.

Для управления состоянием больших приложений существует официальная библиотека Vuex. Vuex позволяет централизованно управлять состоянием приложения с помощью хранилища (store), мутаций (mutations), действий (actions) и геттеров (getters). Это облегчает отслеживание изменений состояния и делает код более организованным и модульным.

Vue компоненты проходят через несколько стадий жизненного цикла, что позволяет разработчикам выполнять код на различных этапах. Основные методы жизненного цикла включают:

- BeforeCreate(): вызывается до создания экземпляра компонента;
- Created(): вызывается после создания экземпляра компонента;
- BeforeMount(): вызывается перед монтированием компонента в DOM;
- Mounted(): вызывается после монтирования компонента в DOM;
- BeforeUpdate(): вызывается перед обновлением компонента;
- Updated(): вызывается после обновления компонента;
- BeforeDestroy(): вызывается перед уничтожением компонента;
- Destroyed(): вызывается после уничтожения компонента.

Из основных преимуществ можно выделить:

- Простота изучения и использования благодаря интуитивно понятному синтаксису и хорошей документации;
- Высокая производительность благодаря легковесной архитектуре и эффективной системе обновления DOM;
- Гибкость и модульность, позволяющие легко интегрироваться с другими библиотеками и фреймворками.

Из недостатков можно выделить:

- Относительно небольшое сообщество по сравнению с React и Angular, что может ограничивать доступность готовых решений и библиотек;

- Ограниченная поддержка крупных корпоративных проектов по сравнению с Angular, который разрабатывается и поддерживается Google.

В заключение, Vue является мощным и гибким инструментом для создания современных веб-приложений благодаря своей реактивной системе данных, компонентному подходу и простоте интеграции. Он позволяет быстро и эффективно создавать сложные пользовательские интерфейсы, что делает его популярным выбором среди разработчиков различных уровней опыта.

2.3.4 Svelte

Svelte — это современный JavaScript-фреймворк, представленный в 2016 году, предназначенный для создания высокопроизводительных пользовательских интерфейсов. В отличие от традиционных фреймворков, таких как React или Vue, Svelte работает на этапе сборки, а не во время выполнения. Это означает, что Svelte компилирует компоненты в высокооптимизированный JavaScript-код, который напрямую манипулирует DOM. Такой подход позволяет значительно уменьшить размер приложения и улучшить его производительность.

Основные особенности Svelte включают:

- Компиляция на этапе сборки: Svelte компилирует компоненты в чистый JavaScript-код, который не требует наличия фреймворка во время выполнения. Это позволяет избежать затрат на виртуальный DOM и снизить размер сборки;
- Реактивные данные: В Svelte используется простая и интуитивная система реактивности, которая позволяет автоматически обновлять интерфейс при изменении состояния. Это упрощает синхронизацию данных и представления, сокращая количество кода, необходимого для управления состоянием;
- Svelte поддерживает локализацию стилей на уровне компонента, что позволяет избегать конфликтов стилей и упрощает поддержку кода. Стили, определенные внутри компонента, применяются только к этому компоненту, что обеспечивает изоляцию и предсказуемость.

Из основных преимуществ можно выделить:

- Высокая производительность: За счет компиляции на этапе сборки и отсутствия виртуального DOM, Svelte позволяет создавать очень быстрые приложения с минимальными задержками;
- Малый размер сборки: Компилированный код Svelte значительно меньше по размеру по сравнению с традиционными фреймворками, что улучшает время загрузки приложения;
- Простота и интуитивность: Svelte использует стандартный синтаксис и предлагает простую модель работы с компонентами, что облегчает его изучение и использование.

Из недостатков можно выделить:

- Меньшее сообщество: По сравнению с более зрелыми фреймворками, такими как React и Angular, сообщество Svelte еще относительно небольшое, что может ограничивать доступность готовых решений и библиотек;
- Ограниченная экосистема: Из-за своей новизны, Svelte имеет менее развитую экосистему инструментов и плагинов по сравнению с другими фреймворками;
- Меньшая поддержка корпоративных проектов: Svelte еще не получил широкого признания среди крупных корпораций, что может ограничивать его использование в масштабных проектах.

Svelte является мощным и инновационным инструментом для создания современных веб-приложений благодаря своей высокоэффективной системе компиляции, простоте использования и легковесности. Он предоставляет разработчикам возможность создавать быстрые и производительные интерфейсы с минимальными усилиями, что делает его привлекательным выбором для многих проектов.

2.3.5 Выбор фреймворка

Все четыре фреймворка — React, Angular, Vue и Svetle — изначально удовлетворяют всем критериям, которые были поставлены в разделе 2.2. Каждый из них обладает необходимыми возможностями для создания современных клиентских веб-приложений, поэтому любой из них мог быть использован для разработки данного проекта.

Однако в разработке было принято решение использовать именно React по нескольким причинам:

- Гибкость: React не задает строгую архитектуру, что позволяет использовать любые библиотеки для решения определенных задач. Это обеспечивает дополнительную гибкость в выстраивании архитектуры приложения;
- Поддержка сообщества: На данный момент React является лидирующим фреймворком среди аналогов, имея самое большое сообщество разработчиков. Это означает наличие большого количества готовых решений, библиотек и инструментов, а также активную поддержку и развитие фреймворка;
- Средняя сложность разработки по сравнению с Angular, что делает React более доступным для изучения и использования.

Основной причиной выбора React стала именно его гибкость, что позволяет адаптировать фреймворк под конкретные требования проекта и использовать его возможности максимально эффективно.

3 РЕАЛИЗАЦИЯ ВЕБ-ИНТЕРФЕЙСА

В данной главе обозреваются все основные библиотеки, которые были использованы, их интеграция в проект, а также описаны основные архитектурные решения, которые были приняты. Кроме того, демонстрируется текущий функционал разработанного веб-интерфейса.

3.1 Основные библиотеки

Для реализации веб-интерфейса приложения был использован ряд библиотек и инструментов, которые помогли решить конкретные задачи. В данной главе рассмотрены основные библиотеки, использованные для создания функциональности приложения, а также объяснен их выбор и интеграция.

3.1.1 Three.js для работы с 3D графикой

В проекте возникла необходимость в использовании библиотеки для 3D графики для создания интерактивных сцен и работы с трехмерными объектами. Для работы с 3D графикой в браузере используется WebGL. WebGL – это программная библиотека для JavaScript, которая позволяет создавать 3D графику, функционирующую в браузерах. Данная библиотека основана на архитектуре OpenGL и использует язык программирования шейдеров GLSL, который имеет C-подобный синтаксис. WebGL интересен тем, что код моделируется непосредственно в браузере. Для этого WebGL использует объект canvas, который был введен в HTML5.

Работа с WebGL и с шейдерами в частности — это довольно трудоемкий процесс. В процессе разработки необходимо описать каждую точку, линию, грань и так далее. Чтобы все это визуализировать, необходимо прописать довольно объемный кусок кода. Для повышения скорости разработки была использована библиотека Three.js. Three.js облегчает работу с 3D-графикой, предоставляя интерфейс высокого уровня абстракции, который скрывает техническую сложность использования WebGL.

Three.js был выбран из-за его универсальности и гибкости, что позволяло создавать сложные сцены и анимации. Основным преимуществом исполь-

зования Three.js является его большое и активное сообщество, которое постоянно поддерживает и развивает библиотеку, обеспечивая большое количество расширений и плагинов, что существенно облегчает разработку [14].

Существуют альтернативы, такие как Babylon.js и PlayCanvas, однако Three.js был предпочтительным выбором благодаря своей гибкости и активному сообществу. Кроме того, библиотека AMI.js, которая в проекте используется для работы с DICOM файлами, в ядре также использует Three.js, что обеспечило дополнительную совместимость и упрощение интеграции.

3.1.2 Ami.js для работы с DICOM файлами

Несмотря на широкие возможности, которые предоставляет Three.js, она не имеет встроенной поддержки работы с файлами DICOM напрямую. Это связано с тем, что формат DICOM является стандартом в медицинской практике и имеет свою специфику. Формат DICOM достаточно сложен и обширен, так как охватывает различные типы данных, используемых в медицинской визуализации, такие как КТ, рентген, УЗИ и другие.

Каждый из этих типов данных реализуется через формат DICOM, но они значительно различаются между собой по структуре и содержимому. Внедрение поддержки DICOM в библиотеку Three.js потребовало бы значительных усилий, учитывая разнообразие и сложность формата. Это, вероятно, является основной причиной отсутствия такой поддержки в Three.js, несмотря на её популярность и широкие возможности. Поэтому пришлось использовать сторонние библиотеки.

Среди библиотек для работы с медицинскими изображениями в формате DICOM, AMI.js выделяется тем, что в его основе лежит Three.js, что обеспечивает эффективную работу с трехмерными визуализациями. Варианты альтернатив, такие как Cornerstone с ядром на базе VTK.js, оказались менее подходящими из-за сложности настройки и менее гибкой архитектуры. Однако использование AMI.js не лишено недостатков. Библиотека не обновлялась с 2018 года и изначально была написана на JavaScript, поддерживая версии Three.js до 0.99.0, что является довольно устаревшей версией.

Также стоит отметить, что AMI.js не имеет поддержки TypeScript, что создавало дополнительные трудности в интеграции с современными проектами, использующими строгую типизацию. В ходе проекта пришлось само-

стоятельно дописывать определения типов для модуля AMI.js. Для написания типизации необходимо было вручную анализировать исходный код библиотеки, что значительно усложняло процесс. В некоторых случаях анализаторы IDE помогали выявить необходимые типы, но зачастую приходилось додумывать их самостоятельно.

В ходе разработки было обнаружено, что при работе со сценой, на которой расположен сагиттальный срез, происходило заметное снижение частоты кадров в секунду до критических уровней, при которых программа фактически зависала. Это особенно заметно проявлялось при высоком разрешении изображаемого среза и большом количестве пикселей.

Этот баг является серьезной проблемой, так как он негативно влияет на производительность приложения, делая его использование затруднительным. Была выдвинута гипотеза о причине этого бага.

Основная гипотеза заключается в том, что проблема связана с отображением воксельной модели. Воксельная модель представляет собой трёхмерный массив данных, где каждый элемент (воксель) содержит информацию о плотности в определенной точке пространства, аналогично тому, как пиксели представляют собой двумерный массив в изображениях.

Для рендеринга воксельных моделей существуют специфичные задачи, связанные с эффективным хранением и доступом к данным. В частности, скорость доступа к данным в различных направлениях в трёхмерном массиве может значительно различаться. Это связано с тем, что расстояние между данными в кэше процессора может варьироваться в зависимости от направления доступа.

Специализированные рендереры для воксельных моделей используют оптимальные методы хранения данных для обеспечения быстрой и эффективной визуализации. Однако, похоже, что в библиотеке AMI.js эти методы либо не были реализованы, либо реализованы недостаточно оптимально, что и приводит к описанным проблемам с производительностью.

На данный момент устранить этот баг не удалось, что ограничивает возможности использования AMI.js для работы с высококачественными медицинскими изображениями. В дальнейшем планируется провести более детальное исследование проблемы и попытаться найти пути ее решения, возможно, с использованием альтернативных библиотек или собственных оптимизаций.

Несмотря на эти минусы и трудности, AMI.js является мощным инструментом для работы с DICOM и предоставляет все необходимые функции для работы с медицинскими изображениями.

3.1.3 Dexie.js для хранения данных

Потребность в личном кабинете доктора возникла из-за необходимости кэшировать данные, загружаемые в приложении. Так как браузерные приложения не имеют прямого доступа к файловой системе, сохранять прогресс работы (например, сегментированные аорты или загруженные снимки) напрямую на компьютер пользователя невозможно. Обновление страницы приводило бы к необходимости заново загружать все данные.

Первоначально предполагалось создать серверную базу данных и серверное приложение для хранения и управления данными. Однако это усложняло бы разработку и требовало создания дополнительных серверных компонентов.

В ходе написания дипломной работы было предложено альтернативное решение: использовать локальную базу данных в браузере для кэширования данных. Индексированная база данных (IndexedDB) позволяет хранить большие объемы данных непосредственно в браузере и предоставляет удобный доступ к ним. Для упрощения работы с IndexedDB была выбрана библиотека Dexie.js.

Dexie.js предоставляет удобный и производительный интерфейс для работы с IndexedDB, позволяя легко сохранять и загружать данные из базы данных. Внедрение Dexie.js позволило перенести кэширование данных с серверной части на клиентскую, что значительно упростило архитектуру приложения и сделало его более независимым от серверных компонентов.

Использование Dexie.js обеспечило следующие преимущества:

- Упрощение серверной части: Серверная часть приложения не зависит от локальных данных и выполняет только те функции, которые основаны на входных данных от клиента;
- Безопасность хранения данных: Данные хранятся на компьютере пользователя и могут быть доступны только с того же домена, на ко-

тором они были сохранены, что делает их безопасными от несанкционированного доступа;

- Производительность и масштабируемость: Dexie.js обеспечивает высокую производительность работы с данными и позволяет масштабировать приложение без значительных изменений в его архитектуре.

Таким образом, использование библиотеки Dexie.js для работы с локальной базой данных в браузере позволило значительно упростить разработку и улучшить функциональность приложения, обеспечив удобное и безопасное кэширование данных на стороне клиента.

Для хранения информации о пациентах и данных медицинских обследований была разработана следующая схема базы данных(Рисунок 3.2).

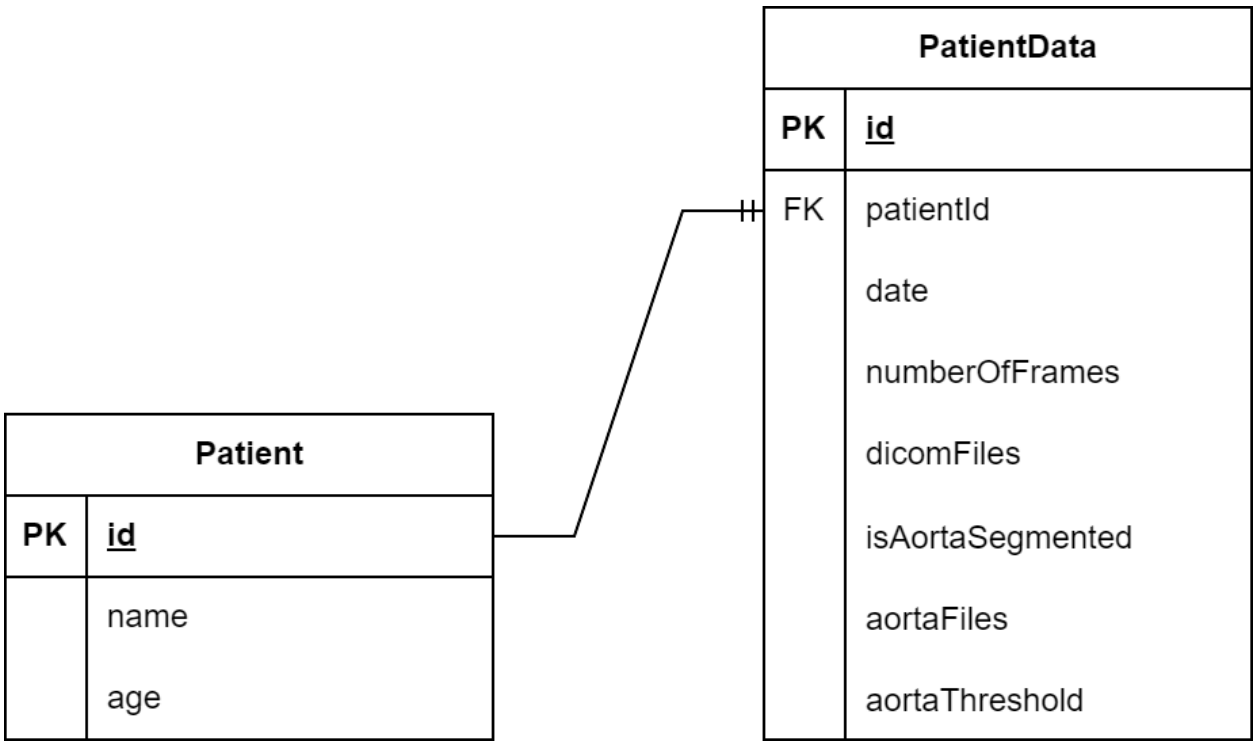

Рисунок 3.1. Схема базы данных

В данной схеме база данных состоит из двух таблиц: Patient и PatientData.

- Таблица Patient содержит основную информацию о пациентах из метаданных;
- Таблица PatientData предназначена для хранения данных, таких как дата исследования, количество кадров, файлы DICOM, данные сегментации.

В будущем планируется модификация данной схемы базы данных для интеграции линий пришиваний и створок клапанов, а также других метаданных пациента при необходимости.

3.1.4 Вспомогательные библиотеки для React

Как было упомянуто в разделе 2.3.1, React сам по себе не является полноценным фреймворком, а скорее библиотекой для создания пользовательских интерфейсов. Для реализации многих ключевых функций требуется использование дополнительных библиотек. В этом разделе мы рассмотрим несколько важных вспомогательных библиотек, которые использовались в проекте.

Для реализации Flux-архитектуры в приложении был выбран Redux. Flux-архитектура помогает управлять состоянием приложения и потоками данных, что особенно важно в больших и сложных приложениях. Redux предоставляет централизованное хранилище для состояния приложения, что позволяет легко управлять состоянием, отслеживать изменения и проводить отладку.

- Централизованное управление состоянием: Все состояние приложения хранится в одном месте, что упрощает управление и отладку;
- Предсказуемость: Изменения состояния осуществляются через специальные механизмы, что делает поведение приложения более предсказуемым;
- Возможность интеграции с различными инструментами: Redux легко интегрируется с различными инструментами для отладки и тестирования.

Создание пользовательского интерфейса с нуля может быть трудоемким и времязатратным процессом, особенно когда речь идет о создании стандартных элементов интерфейса, таких как кнопки, списки, панели навигации и т.д. Для упрощения этого процесса была использована библиотека Material-UI.

Material-UI предоставляет набор готовых компонентов, стилизованных в соответствии с принципами Material Design. Это позволяет:

- Ускорить разработку интерфейса: Использование готовых компонентов экономит время, необходимое на разработку и стилизацию;
- Обеспечить единый стиль приложения: Все компоненты стилизованы в одном стиле, что делает интерфейс приложения более согласованным и профессиональным;
- Поддержка и обновления: Material-UI активно поддерживается сообществом разработчиков и регулярно обновляется, что обеспечивает совместимость с новыми версиями React и других инструментов.

Эти вспомогательные библиотеки в совокупности с React позволили создать мощный и функциональный веб-интерфейс, который удовлетворяет все текущие потребности проекта и обладает потенциалом для дальнейшего развития и улучшения.

3.2 Ключевые моменты реализации

Далее описаны ключевые моменты, связанные с объединением различных библиотек и инструментов в единую архитектурную систему. Проблема заключалась в необходимости интеграции различных библиотек, таких как React и Three.js, и управления их взаимодействием в рамках единого приложения. Также описаны основные аспекты взаимодействия с серверной частью.

3.2.1 Архитектура приложения

Для создания полноценного веб-приложения необходимо было объединить несколько библиотек, каждая из которых выполняла бы свою специфическую задачу. В нашем случае это React для создания пользовательского интерфейса и Three.js для работы с 3D графикой. Для удобной интеграции Three.js и React существует библиотека React-Three, однако ее использование не было возможно, так как для использования AMI.js нужна версия Three.js 0.99.0, а для React-Three нужна версия Three.js от 0.125.0.

Первоначальной идеей было взять основные подходы из React-Three и реализовать их самостоятельно для нашего приложения, однако возникли трудности.

Парадигма React-Three хорошо подходила для реализации одной сцены или множества независимых сцен, однако у нас все четыре сцены сильно зависели друг от друга, что создавало сложности в реализации такой парадигмы. Работать с изменяемыми объектами в React и Three.js, а также управлять состоянием и взаимодействием между зависимыми сценами, оказалось непросто.

Одной из проблем, с которой пришлось столкнуться при использовании React, является его строгий режим (strict mode). Строгий режим важен для разработки, так как помогает отлавливать потенциальные проблемы и побочные эффекты, отрисовывая каждый компонент дважды. Это позволяет React выявить различные ошибки и некорректное поведение на ранних стадиях. Однако использование строгого режима приводило к дублированию некоторых объектов, что создавало проблемы при работе с 3D сценами.

Подход к архитектуре приложения претерпевал значительные изменения на протяжении разработки. В конечном итоге была выбрана парадигма, которая заключалась в инкапсуляции логики работы с 3D сценой и пользовательским интерфейсом. Для их связи был создан удобный API, позволяющий выполнять операции с 3D сценой (например, добавление или удаление элементов) из пользовательского интерфейса.

Основная идея заключалась в инкапсуляции всей логики работы с 3D объектами в специальных классах. Были созданы два ключевых класса: Renderer3D и Renderer2D для работы с трехмерными и двухмерными сценами соответственно. В этих классах реализованы основные методы, такие как создание сцены, добавление и удаление элементов, а также методы для управления сценами и их визуализацией.Как упоминалось ранее в разделе 3.1.1, для создания сцены нужен элемент canvas из HTML. Однако, чтобы сделать эти классы полностью самостоятельными и независимыми от внешних условий при создании, было решено порождать HTML элемент canvas непосредственно внутри класса, вызывая соответствующие функции для создания HTML элементов, а затем вставлять его в компоненты React в нужном месте.

Эти классы включают основные свойства, описывающие сцену, такие как сама сцена, камера, рендерер и прочие элементы. Различия между классами 3D и 2D сцен заключаются в типах используемых камер и проекциях.

- Перспективная камера используется для 3D сцены. Это камера, которая имитирует реальное восприятие объектов, так, как они выглядят в реальной жизни;
- Ортографическая камера используется для 2D сцены. Она отображает ортогональную проекцию предметов на плоскость, где находится камера, что позволяет получать плоские изображения без перспективных искажений.

Так как DICOM файлы представляют собой трехмерные объекты, состоящие из множества двумерных снимков, для взаимодействия с ними используется комбинация наложения этих снимков друг на друга, что проецирует 3D картинку. Эти задачи решаются с помощью Stack Helper и Localizer Helper, предоставляемых библиотекой AMI.js.

Поскольку сцены и отображаемые объекты являются тяжелыми и ресурсоемкими, важно было избежать их дублирования. Для этого был применен паттерн Singleton, который гарантировал, что каждая сцена и объект создаются единожды и могут быть использованы повторно в разных частях приложения. Это позволило решить проблему дублирования объектов, возникающую из-за использования strict mode в React.

Схема такой архитектуры показана на диаграмме компонентов (Рисунок 3.2).

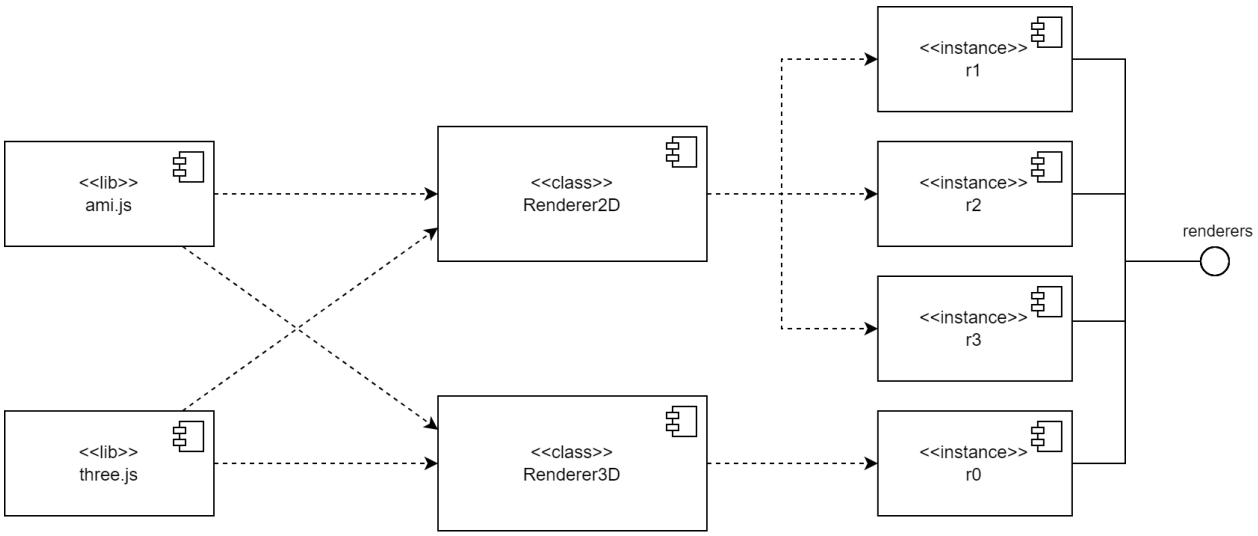

Рисунок 3.2. Диаграмма компонентов логики работы с 3D объектами

Для удобного доступа к рендерерам в компонентах React были созданы хуки, такие как useRenderer и useRenderers. Эти хуки обеспечивают доступ к

рендерерам из любого компонента дерева компонентов, позволяя удобно и эффективно работать с 3D и 2D сценами. Был реализован хук для доступа как ко всем рендерарам, так и к текущему. На рисунке 3.3 это изображено схематично в диаграмме компонентов.

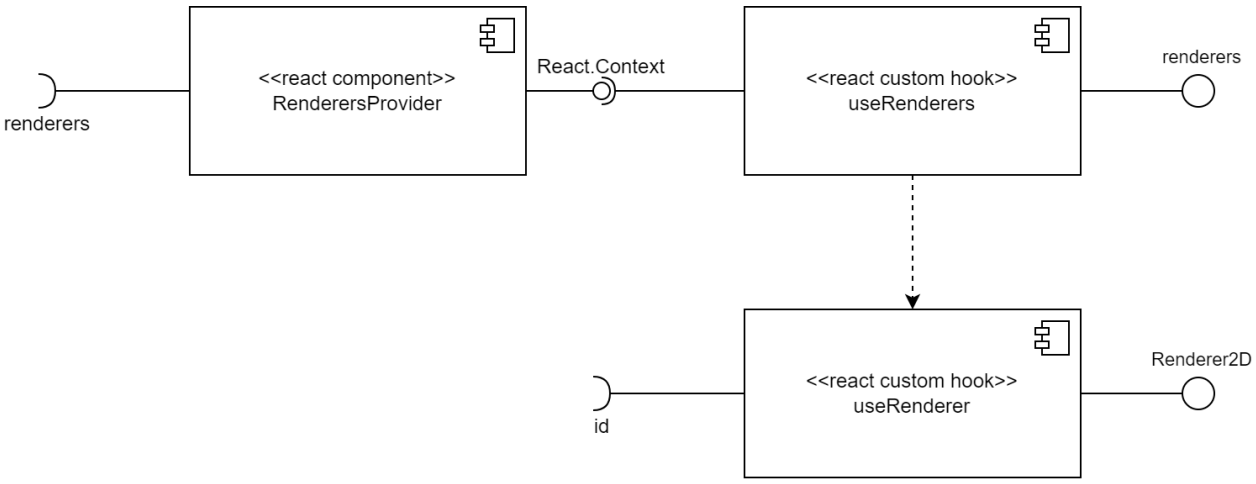

Рисунок 3.3. Диаграмма компонентов доступа рендереров через хуки в UI

3.2.2 Компоненты React

Условно все компоненты можно разделить на три группы:

- Компоненты, связанные со сценой;
- Провайдеры;
- UI-компоненты.

Первые отвечают за инициализацию и управление 3D-сценой, а также за создание и отображение 3D-объектов. Они взаимодействуют с классами Renderer2D и Renderer3D для управления сценами. Список компонентов:

- Canvas2D: Компонент для инициализации 2D сцены;
- Canvas3D: Компонент для инициализации 3D сцены;
- LocalizerHelper: Компонент для работы с локализатором;
- StackHelper: Компонент для работы со стеком DICOM;
- Aorta: Компонент для отображения сегментированной аорты.

Провайдеры обеспечивают управление и доступ к основным элементам системы. Они служат для передачи данных и методов через дерево компонентов, обеспечивая инкапсуляцию и удобное использование ресурсов:

- AortaProvider: Обеспечивает доступ к данным сегментированной аорты;
- QuadViewProvider: Определяет, какая из сцен используется и управляет четырьмя различными видами;
- RenderersProvider: Обеспечивает доступ к классам рендереров для работы с 2D и 3D сценами;
- StackProvider: Обеспечивает доступ к DICOM снимкам и управляет их состоянием.

UI-компоненты представляют собой элементы пользовательского интерфейса, которые обеспечивают взаимодействие пользователя с системой. Они включают в себя навигационные элементы, списки пациентов и карточки данных пациента:

- AppToolbar: Верхняя панель навигации приложения;
- AppToolbarButton: Кнопки на верхней панели навигации;
- PatientCard: Карточка данных пациента, отображающая информацию о конкретном пациенте;
- PatientCardList: Список карточек пациентов, содержащий данные обо всех пациентах;
- PatientDataCard: Карточка данных пациента, отображающая информацию о конкретном снимке;
- PatientDataCardList: Список карточек данных пациента, содержащий информацию обо всех снимках пациента;
- QuadView: Компонент, объединяющий все четыре сцены.

3.2.3 Взаимодействие с серверной частью

Общение между клиентом и сервером было реализовано через WebSocket. Для этого было несколько причин:

- Необходимость отслеживания длительных операций: Процесс выполнения некоторых операций может занимать значительное время. Например, сегментирование аорты выполняется около 30-40 секунд, а моделирование до 3 минут. HTTPS запросы для таких длительных

операций неэффективны, так как они не позволяют эффективно отслеживать прогресс и могут приводить к таймаутам. WebSocket позволяет поддерживать постоянное соединение, обеспечивая двустороннюю связь между клиентом и сервером, что позволяет эффективно отслеживать прогресс выполнения операций;

- Совместимость с Docker-контейнерами: Все решение, включающее все модули, будет упаковано в Docker-контейнер и доступ к серверу и клиенту будет осуществляться с одного и того же домена. Использование HTTPS запросов в данном случае вызывало бы ошибки в браузере из-за политик безопасности, связанных с кросс-доменными запросами. WebSocket позволяет безопасно обходить эти ограничения, обеспечивая надежное взаимодействие между клиентом и сервером;
- Опыт использования WebSocket в предыдущей версии проекта: взаимодействие с сервером также осуществлялось через WebSocket по аналогичным причинам. Уже была готова некоторая база кода для работы с WebSocket, что облегчило внедрение этой технологии в текущий проект.

Было создано API для взаимодействия с сервером, упакованное в отдельный пакет. Это API предоставляет интерфейс для отправки и получения данных через WebSocket. API включает методы для отправки снимков и параметров сегментации (Threshold) на сервер, а также для обработки сообщений, поступающих от сервера. В рамках проекта была проведена работа по реализации типизации для этого API.

Механизм взаимодействия через WebSocket включает следующие шаги:

1. Отправка данных на сервер: Клиент отправляет снимки и порог сегментации на сервер через WebSocket;
2. Обработка сообщений от сервера: Сервер может отправлять сообщения о прогрессе выполнения операции. Когда сервер отправляет сообщение, в API создается искусственное событие;
3. Подписка на события: на клиенте создается подписка на эти искусственные события с помощью стандартных Event Listener в JavaScript. Например, при получении сообщения о прогрессе сегментации клиент обновляет прогресс бар;

4. Получение результатов: Когда сервер завершает операцию и отправляет результат (например, STL файл аорты), соответствующее событие срабатывает, и клиент обрабатывает полученные данные.

Схематично взаимодействие сервера и клиента отражено на рисунке 3.4.

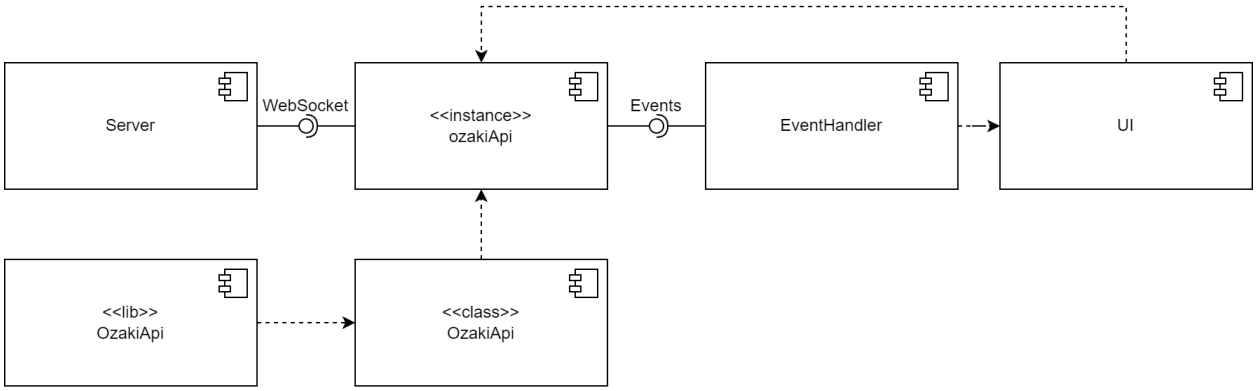

Рисунок 3.4. Диаграмма компонентов взаимодействия серверной и клиентской части

STL файл сохраняется в локальную базу данных (Dexie.js) для последующего использования. Для загрузки STL файлов используется модуль Three.js, который обеспечивает визуализацию и дальнейшую работу с 3D моделями.

Таким образом, использование WebSocket для взаимодействия с сервером обеспечило эффективное и надежное выполнение длительных операций, а также позволило реализовать удобный механизм отслеживания прогресса и получения результатов в реальном времени.

Одной из ключевых задач при развертывании клиент-серверного приложения было автоматическое определение URL сервера в production режиме, когда клиент и сервер находятся в одном Docker-контейнере. Для этого была реализована соответствующая функция, которая автоматически определяет URL сервера.

Когда клиентская версия находится в developer режиме, автоматическое определение URL сервера необходимо отключать. В этом режиме удобно вручную вводить нужный URL для тестирования и отладки. Чтобы автоматизировать этот процесс и избежать необходимости ручного переключения URL, использовалась одна из функций сборщика Vite.

Vite позволяет настраивать окружение и переменные окружения, которые фиксируют текущий режим работы (developer или production). В зависимости от режима, в код вставляется соответствующая переменная.

Была настроена система переменных окружения, которая позволяет автоматически переключаться между режимами developer и production. Это существенно упростило процесс разработки и развертывания приложения, так как отпала необходимость вручную править код.

3.3 Демонстрация работы приложения

В этом разделе представлены скриншоты работающего приложения, демонстрирующие основные функции и возможности системы. Визуальные примеры помогают лучше понять, как реализованы и функционируют различные компоненты приложения.

На рисунке 3.5 показано отображение DICOM снимков на всех четырех сценах. Это позволяет пользователям просматривать снимки пациента в разных проекциях и настраивать параметры визуализации для получения наиболее информативных изображений.

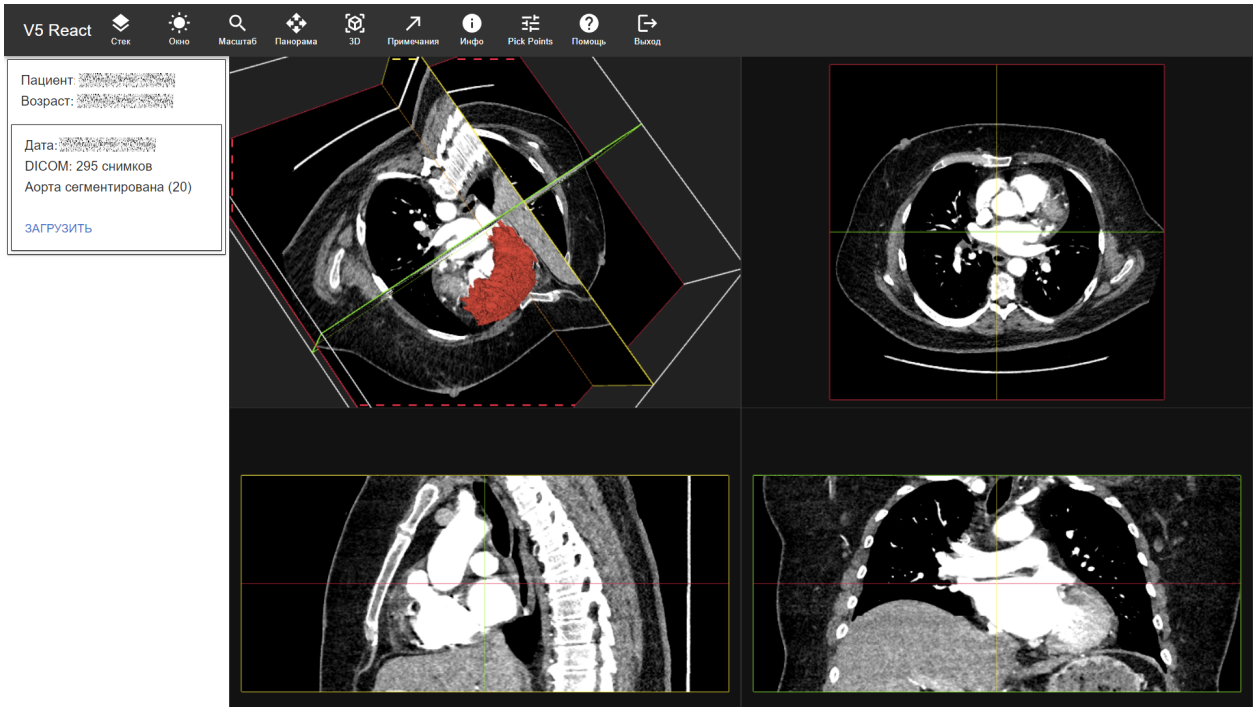

Рисунок 3.5. КТ пациента в 3 проекциях

На рисунке 3.6 демонстрируется сегментированная аорта для пациента. Сегментация позволяет выделить аорту на снимках и визуализировать ее в 3D пространстве.

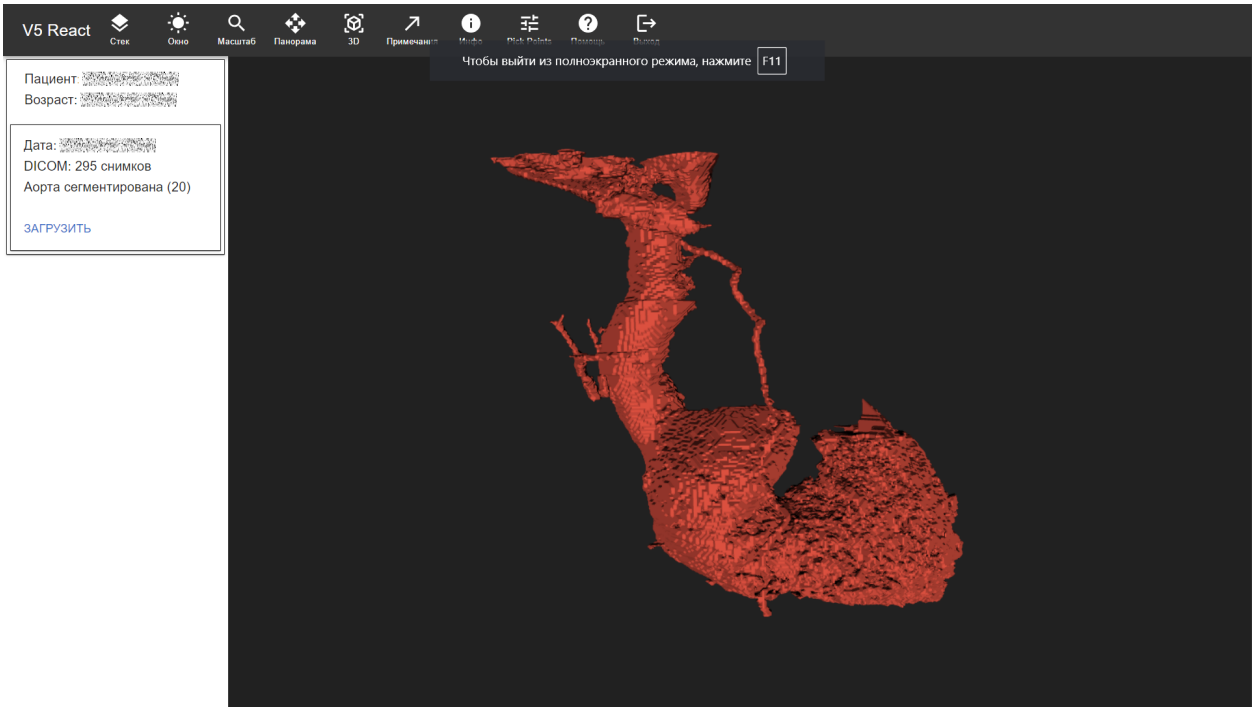

Рисунок 3.6. Сегментировная 3D модель аорты

На рисунке 3.7 демонстрируется диалог для сегментации аорты.

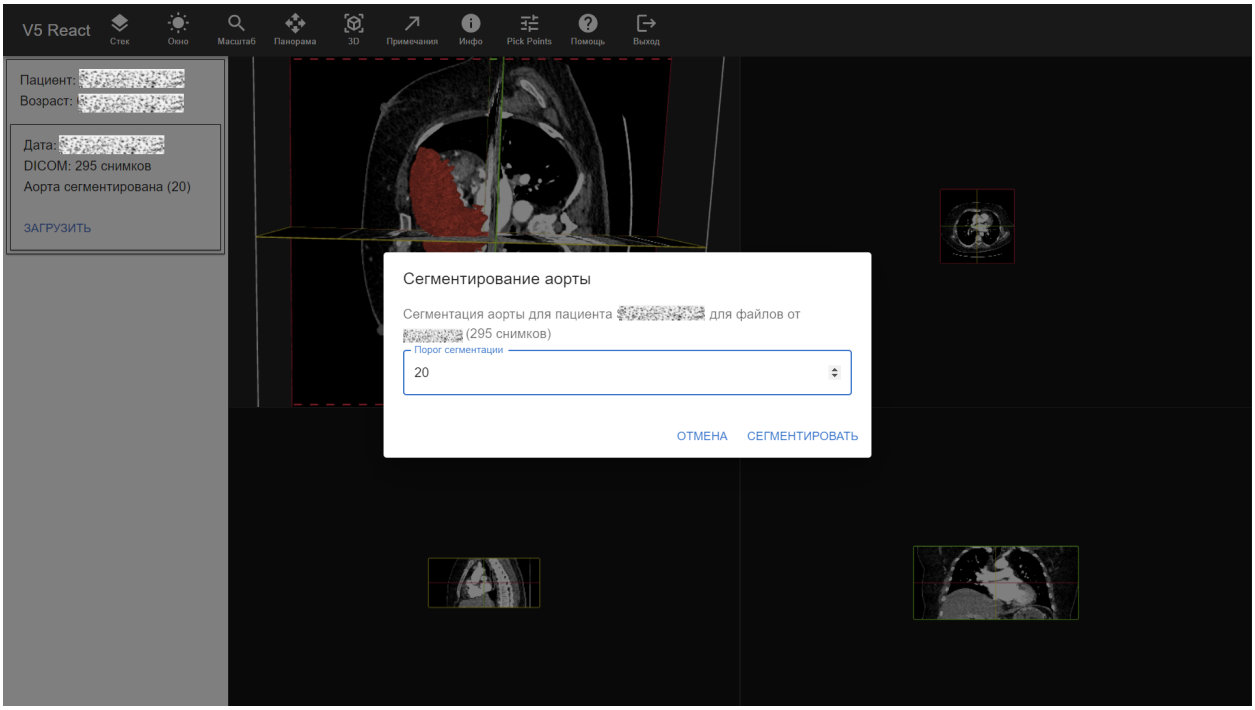

Рисунок 3.7. Диалог сегментации аорты

На рисунке 3.8 демонстрируется отображение прогресса при выполнении сегментации на стороне сервера, что позволяет контролировать и отслеживать выполнение длительных операций.

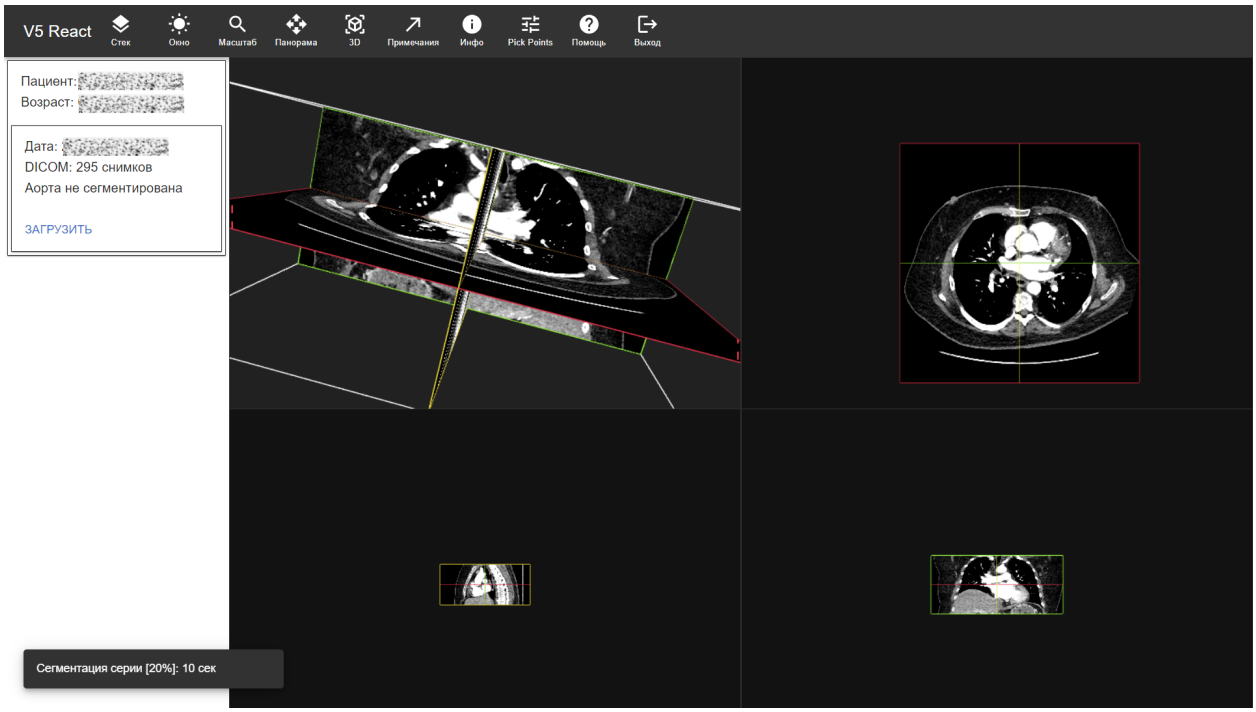

Рисунок 3.8. Прогресс сегментации

На данном этапе удалось реализовать только две из четырех основных функций, описанных в разделе разделе 1.3. В частности, были разработаны интерфейсы для загрузки данных пациента и их просмотра, а также сегментация и визуализация аорты.

Разработка следующих функций, таких как проведение линий приживания створок клапана аорты и визуализация результатов моделирования, будет следующим шагом в развитии проекта. Эти функции планируется реализовать в рамках последующих этапов работы.

Заключение

Целью данной работы была разработка пользовательского веб-интерфейса в системе поддержки принятия врачебных решений для операции Озаки с использованием современных фреймворков. В ходе выполнения работы цель была достигнута. Был создан веб-интерфейс, реализующий следующие ключевые функции:

- Загрузка данных пациента и их просмотр: пользователи могут легко загружать КТ данные пациент в формате DICOM, а также просматривать их, используя основные функции такие как панорамирование, зумирование и вращение изображений.
- Сегментация и визуализация аорты: обеспечена интеграция с серверной частью для выполнения задач по сегментации аорты и наглядного отображения результатов сегментации.

Разработанный веб-интерфейс основывается на современных инструментах, которые были выбраны путем анализа проблем старой кодовой базы и современных подходов к созданию веб-интерфейсов. Интерфейс реализован на языке TypeScript, использует сборщик Vite и фреймворк React. Для работы с 3D-графикой и медицинскими изображениями применяются библиотеки Three.js и Ami.js. Он соответствует современным требованиям и критериям, таким как масштабируемость, развертываемость, поддерживаемость и сопровождаемость.

Работа над проектом будет продолжена в рамках аспирантуры. Вторая часть разработки включает проведение линий пришивания створок клапана аорты, что станет следующим этапом совершенствования системы.

Код проекта доступен на github https://github.com/Micro-ice-ice/v5-react.

Список использованных источников

1. *Василевский Ю.В.*, *Симаков С.С.*, *Гамилов Т.М.* [и др.]. Персонализация математических моделей в кардиологии: трудности и перспективы // Компьютерные исследования и моделирование. — 2022. — Т. 14, № 4. — С. 911—930.
2. *Shaalan D.*, *Jusoh S.* Visualization in medical system interfaces: UX guidelines // 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). — IEEE. 2020. — P. 1–8.
3. *Pirola S.*, *Mastroiacovo G.*, *Bisleri G.*, *Polvani G.* Ozaki procedure: how I teach it // The Annals of Thoracic Surgery. — 2021. — Vol. 111, no. 6. — P. 1763–1769.
4. *Alhan C.* Ozaki procedure // Turkish Journal of Thoracic and Cardiovascular Surgery. — 2019. — Vol. 27, no. 4. — P. 451.
5. The Society of Thoracic Surgeons. — 2023. — URL: https://ctsurgerypatients.org/adult-heart-disease/aortic-valve-disease (visited on 06/07/2024).
6. *Nishimura R.A.* Aortic valve disease // Circulation. — 2002. — Vol. 106, no. 7. — P. 770–772.
7. *Чернов И.*, *Энгиноев С.*, *Комаров Р.* [и др.]. Непосредственные результаты операции Ozaki: многоцентровое исследование // Российский кардиологический журнал. — 2020. — S4. — С. 13—18.
8. *Pirola S.*, *Mastroiacovo G.*, *Arlati F.G.*, [et al.]. Single center five years' experience of Ozaki procedure: midterm follow-up // The Annals of thoracic surgery. — 2021. — Vol. 111, no. 6. — P. 1937–1943.
9. *Ozaki S.* Ozaki Procedure: 1,100 patients with up to 12 years of follow-up // Turkish Journal of Thoracic and Cardiovascular Surgery. — 2019. — Vol. 27, no. 4. — P. 454.
10. *Salamatova V.Y.*, *Liogky A.A.*, *Karavaikin P.A.*, [et al.]. Numerical assessment of coaptation for auto-pericardium based aortic valve cusps // Russian Journal of Numerical Analysis and Mathematical Modelling. — 2019. — Vol. 34, no. 5. — P. 277–287.

11. *Аверина Т.Б.* Искусственное кровообращение // Анналы хирургии. — 2013. — № 2. — С. 5—12.

12. *Байдыбеков А.*, *Гильванов Р.*, *Молодкин И.* Современные фреймворки для разработки web-приложений // Интеллектуальные технологии на транспорте. — 2020. — 4 (24). — С. 23—29.

13. *Saks E.* JavaScript Frameworks: Angular vs React vs Vue. — 2019.

14. *Dirksen J.* [et al.]. Three. js essentials. — Packt Publishing, 2014.