мягких тканей поясничного отдела позвоночника человека применяется метод конечных элементов [3], поскольку в поставленной задаче имеет место сложная геометрическая структура. Полученные результаты позволяют провести биомеханический анализ заданного отдела позвоночника. Предлагаемая методика позволит получить новые знания механических свойств поясничного отдела позвоночника человека, моделировать новые инструментарии для хирургического лечения и прогнозировать результаты их применения.

Литература

1. Семенченя, Т. С. Construction of an individual geometric 3D model of the lumbar spine of a person based on the analysis of medical images / Т. С. Семенченя, К. С. Курочка // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2020) : сборник научных трудов / Белорусский государственный университет информатики и радиоэлектроники ; редкол.: В. В. Голенков (гл. ред.) [и др.]. – Минск, 2020. – Вып. 4. – С. 291–296.
2. Курочка, К. С. Алгоритм построения межпозвонковых дисков на основе КТ-изображений / Т. С. Семенченя, К. С. Курочка // Молодежная наука: вызовы и перспективы : материалы IV Международной научно-практической конференции студентов, аспирантов и молодых ученых, 8 апреля 2021 г., Макеевка : в 11 т. Том 3 / ГОУ ВПО «Донбасская аграрная академия». – Макеевка : ДОНАГРА, 2021. – Т. X. – С. 169–171.
3. Сегерлинд, Л. Применение метода конечных элементов. / Л. Сегерлинд. – Москва : Мир, 1979. – 392 с.

Е. А. Сивцов
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **Е. И. Сукач**, канд. техн. наук, доцент

ПРИНЦИПЫ ВЗАИМОДЕЙСТВИЯ ОСНОВНЫХ КОМПОНЕНТОВ ВЕБ-ПРИЛОЖЕНИЯ «ВИДЕО-СЕРВИС»

В основе веб-приложения «Видео-сервис» лежит стандартный принцип взаимодействия сервера и клиента, однако, с некоторыми отличиями и нововведениями относительно разработки рядовой клиент-

серверной модели. При проектировании системы приложения был использован такой архитектурный стиль взаимодействия компонентов распределенного приложения в сети, как REST («передача состояния представления»). REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной медиа системы. В общем случае, REST является простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL (унифицированный указатель ресурса), а каждый URL в свою очередь, имеет строго заданный формат. Использование данного архитектурного стиля позволило справиться с возможными проблемами производительности и привести общую архитектуру приложения к более простому для понимания виду.

Положительный эффект заключается в применении ряда конкретных концепций, характерных для REST архитектуры: передача данных вместо представлений, унифицированные интерфейсы конечных точек, взаимодействие без сохранения состояния, опциональное кэширование. Далее, рассмотрим эти концепции с примерами их имплементации в самом приложении [1, 2].

И так, почему же при разработке данного приложения лучше передавать данные нежели представления? Потому что всю работу по визуальной части приложения берет на себя библиотека React. Именно она манипулирует графическими элементам, анимациями, видеоплеерами, выводит на экраны браузеров пользователей динамический и интерактивный контент, поэтому серверу нет нужды вторгаться в этот процесс и тем более тратить свои драгоценные ресурсы на задачи, которые он не обязан выполнять. К дополнительным плюсам можно отнести то, что таким образом улучшается понимание системы разработчиком, так как изначально понятно какой модуль в какой системе выполняет тот или иной функционал, например: модуль «Видео» в части пользовательского интерфейса будет отображать все визуальные компоненты, для взаимодействия пользователя с видео функционалом и модуль с таким же названием на сервере будет предоставлять функции для взаимодействия лишь с данными о видео.

Чем полезны в данном случае унифицированные интерфейсы? Они выступают в роли, своего рода мостов связи между сервером и клиентом и помогают определить один едный стандартный способ общения путем передачи запросов и принятия ответов – следование подобной концепции при разработке практически любого проекта априори не может быть плохим решением. В приложении, в качестве

унифицированных интерфейсов используются конечные точки на сервере и комбинации заранее определенных статических и динамических URL-адресов. Конечные точки – это стандартизованные функции, которые будут вызваны, когда на сервер придет HTTP-запрос для URL-адресса, за которым зарегестрирована непосредственная функция. После обработки запроса такая функция может передать ответ другой подобной функции по цепочке или же возвратить результат в качестве HTTP-ответа. Результат может содержать какие-либо данные в виде сообщений или же объектов, либо, в случае получения информации о видео, ответом может быть открытый поток для чтения видео файла по частям, ведь передавать в ответе многовесовые целые видеофайлы попросту неэффективно, а для подобных платформ, оперирующих большим объемом данных, это попросту невозможно.

Почему было выбрано взаимодействие без сохранения состояния? Так как приложение обрабатывает большой поток данных: входящие пользовательские видео, исходящий видео-трафик, то хранение еще и состояния сессий пользователей или каки-либо промежуточных данных, однозначно сказывается на производительности и общему проценту успешности обработки запросов сервером. К тому же, в REST архитектуре сервер и пользователь стандартно являются отдельными сущностями, клиент посылает запрос, сервер его обрабатывает и посылает ответ, поэтому сохранение состояния в круговороте запросов и ответов ставит под сомнение выбор большинства предшедствующих концепций.

Как реализовано использование опционального кэширования? REST-архитектура предусматривает возможность кэширования, путем установки заголовков при отправке ответа, однако в данном приложение используется немного иная концепция кэширования. В браузере есть так называемое локальное хранилище, которое сохраняет данные в виде «ключ/значение» даже после закрытия браузера пользователем. Это хранилище целиком и полностью управляется клиентской частью системы, однако некоторые данные оно получает из ответов на определенные запросы сервера. Например: после успешной авторизации пользователя сервер возвращает зашифрованный временно валидный объект, уникально характеризующий текущего пользователя, так называемый токен. Этот объект сохраняется в локальном хранилище браузера, чтобы при последующих запросах сервер предоставлял доступ к закрытым для не авторизованных пользователей ресурсам, путем проверки токена на валидность.

В заключении, можно сказать, что строгое следование архитектурным концепциям не всегда может положительно влиять на результат.

Некоторые проекты требуют уникальных решений в процессе самой разработки и безоговорочное следование тем или иным концепциям может выступать обоюдоострым мечом на войне за успех проекта. Тем не менее, в данном веб-приложении, вышеперечисленные концепции применяются и реализуются в полной мере без какого-либо ущерба.

Литература

1. Арно, Л. Проектирование веб-API / Л. Арно. –М. : ДМК-Пресс, 2020. – 440 с.

2. Mozilla Foundation, Веб-технологии для разработчиков [Электронный ресурс] / Resources for Developers, by Developers. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web. – Дата доступа: 03.03.2023.

Н. Е. Сидоров
(ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **В. А. Дробышевский**, ст. преподаватель

РЕАЛИЗАЦИЯ АРХИТЕКТУРЫ ВЕБ-РЕСУРСА ДЛЯ СОЗДАНИЯ И ПРОВЕДЕНИЯ СОЦИОЛОГИЧЕСКИХ ОПРОСОВ

Архитектура приложения описывает в себе четко изложенный структурный принцип, по которому создано приложение. Веб-ресурсу для создания и проведения социологических опросов в Учреждении «Территориальный центр социального обслуживания населения Ленинского района г.Бобруйска» были поставлены персональные задачи, исходя из которых была выбрана клиент-серверная архитектура.

Клиент-серверная архитектура хорошо подходит для опросника, так как пользователь обращается к программе, а она в свою очередь обрабатывает данные на стороне клиента и передаёт их на сервер, обращаясь непосредственно к самой базе данных. Архитектура работы приложения представлена на рисунке 1.

Архитектура работы приложения разделяет две независимые роли клиента и сервера.

Клиент отвечает за наглядный вывод данных и прямое взаимодействие с пользователем (участником опроса, социологом).