МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра дискретной математики и информационных технологий

РАЗРАБОТКА И АНАЛИЗ ЛЕГКОВЕСНОГО ПРОТОКОЛА ОБМЕНА ДАННЫМИ ДЛЯ ВЕБ-ПРИЛОЖЕНИЯ 'ЦИФРОВАЯ АПТЕЧКА' НА СТЕКЕ NODE.JS/REACT.

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 421 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Мухачёва Владислава Сергеевича

Научный руководитель
ассистент _____ Е. А. Синельников

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире веб-приложения играют все более значимую роль в различных сферах жизни, включая здравоохранение и самообслуживание. Приложения, предназначенные для управления личными данными, такими как информация о лекарственных препаратах в домашней аптечке, требуют не только удобного пользовательского интерфейса, но и эффективного взаимодействия между клиентской и серверной частями. Особую актуальность это приобретает в контексте мобильных устройств и нестабильных интернет-соединений, где объем передаваемых данных и скорость их обработки напрямую влияют на пользовательский опыт и доступность сервиса.

Стандартным де-факто для обмена данными в веб-приложениях является формат JSON (JavaScript Object Notation). Благодаря своей текстовой природе и простоте интеграции с JavaScript, JSON получил широкое распространение. Однако его текстовый формат, включающий полные имена ключей и строковое представление всех данных, часто приводит к избыточности передаваемой информации. Эта избыточность может становиться критичной для приложений, ориентированных на экономию трафика и высокую скорость отклика, особенно в медицинских приложениях, где своевременный доступ к информации может быть важен, а условия сетевого подключения не всегда оптимальны.

Для решения проблемы избыточности JSON были разработаны различные бинарные форматы сериализации, такие как MessagePack, Protocol Buffers, Avro и другие. Они предлагают более компактное представление данных за счет использования бинарных кодировок для типов данных и числовых идентификаторов для ключей. Однако, применение универсальных бинарных форматов не всегда является оптимальным решением, так как они могут вносить дополнительную сложность в разработку или не полностью учитывать специфику конкретного приложения и его наборов данных.

В связи с этим, актуальной задачей является исследование и разработка специализированных, легковесных протоколов или форматов обмена данными, адаптированных под конкретные нужды веб-приложений. Такой подход может позволить достичь значительного снижения объема трафика и потенциального ускорения процессов (де)сериализации по сравнению со стандартным JSON, а также, возможно, превзойти по некоторым показателям

универсальные бинарные форматы в рамках узкоспециализированной задачи.

Целью данной дипломной работы является разработка и анализ эффективности собственного легковесного протокола обмена данными, предназначенного для использования в веб-приложении "Цифровая аптечка реализованном на стеке Node.js (Express) и React.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих подходов к сериализации данных в веб-приложениях, выявить преимущества и недостатки формата JSON и популярных бинарных альтернатив.
2. Сформулировать требования к легковесному протоколу обмена данными для веб-приложения "Цифровая аптечка учитывая специфику передаваемых данных (информация о лекарствах, напоминаниях, истории использования).
3. Разработать концепцию и спецификацию собственного легковесного протокола/формата сериализации данных, ориентированного на минимизацию трафика и потенциальное ускорение обработки.
4. Реализовать функции сериализации и десериализации для разработанного протокола на стороне сервера (Node.js) и клиента (JavaScript).
5. Разработать веб-приложение "Цифровая аптечка использующее как стандартный JSON, так и разработанный протокол для обмена данными с бэкендом, с возможностью переключения между ними.
6. Интегрировать MessagePack в приложение для трехстороннего сравнения.
7. Провести сравнительный анализ эффективности разработанного протокола с JSON и MessagePack по метрикам: объем передаваемых данных и, по возможности, скорость (де)сериализации на реальных сценариях использования веб-приложения.
8. Обосновать целесообразность применения предложенного решения для приложений с аналогичными требованиями к эффективности обмена данными.

1 Теоретические основы и анализ существующих решений для обмена данными в веб-приложениях

Современные веб-приложения представляют собой сложные системы, взаимодействие компонентов которых часто построено на основе клиент-серверной архитектуры. Эффективность и производительность таких приложений во многом зависят от способов организации обмена данными между клиентом и сервером. В данной главе будут рассмотрены ключевые принципы этого взаимодействия, а также проведен анализ наиболее распространенных форматов сериализации данных, их преимуществ и недостатков.

1.1 Принципы клиент-серверного взаимодействия в веб-приложениях

Клиент-серверная архитектура является доминирующей моделью для построения большинства сетевых приложений, включая веб-приложения. В этой модели можно выделить две основные взаимодействующие стороны:

— Клиент – программа или устройство, которое инициирует запросы к серверу для получения каких-либо ресурсов или услуг. В контексте веб-приложений клиентом чаще всего выступает веб-браузер пользователя или мобильное приложение. Клиент отвечает за пользовательский интерфейс и отображение данных, полученных от сервера.

— Сервер – мощный компьютер или программный комплекс, который ожидает и обрабатывает запросы от клиентов, предоставляя им доступ к данным, выполняя бизнес-логику и управляя ресурсами. Веб-серверы, серверы приложений и серверы баз данных являются типичными примерами серверных компонентов.

Взаимодействие между клиентом и сервером в веб-среде преимущественно осуществляется с использованием протокола передачи гипертекста HTTP (HyperText Transfer Protocol) или его защищенной версии HTTPS (HTTP Secure). HTTP является протоколом прикладного уровня, который определяет набор правил и соглашений для обмена сообщениями между клиентом и сервером.

Ключевыми элементами HTTP-взаимодействия являются:

— Запрос (Request): Сообщение, отправляемое клиентом серверу. Запрос содержит:
  – Метод запроса (Request Method): Указывает на желаемое действие

над ресурсом (например, GET для получения данных, POST для отправки данных на сервер для создания ресурса, PUT для обновления существующего ресурса, DELETE для удаления ресурса).
  - URL (Uniform Resource Locator): Адрес запрашиваемого ресурса на сервере.
  - Версия протокола HTTP.
  - Заголовки запроса (Request Headers): Дополнительная информация о запросе, такая как тип принимаемого контента (Accept), тип отправляемого контента (Content-Type), информация для аутентификации и кэширования.
  - Тело запроса (Request Body): Опциональная часть, содержащая данные, передаваемые на сервер (например, данные формы при POST-запросе).
- Ответ (Response): Сообщение, отправляемое сервером клиенту в ответ на его запрос. Ответ содержит:
  - Версия протокола HTTP.
  - Код состояния (Status Code): Трехзначное число, указывающее на результат обработки запроса (например, 200 OK – успешный запрос, 404 Not Found – ресурс не найден, 500 Internal Server Error – внутренняя ошибка сервера).
  - Сообщение состояния (Reason Phrase): Краткое текстовое описание кода состояния.
  - Заголовки ответа (Response Headers): Дополнительная информация об ответе, такая как тип контента в теле ответа (Content-Type), информация о кэшировании, дата ответа.
  - Тело ответа (Response Body): Опциональная часть, содержащая запрошенные данные или информацию об ошибке.

Для структурированного взаимодействия между различными программными компонентами, в том числе между клиентом и сервером, используются API (Application Programming Interface). В контексте веб-сервисов часто говорят о Web API, которые определяют набор эндпоинтов (URL), доступных методов HTTP, форматов запросов и ответов. REST (Representational State Transfer) является одним из наиболее популярных архитектурных стилей для построения Web API, использующим стандартные методы HTTP и

часто JSON в качестве формата данных.

1.2 Форматы сериализации данных: обзор и характеристики

Для обмена структурированными данными между клиентом и сервером необходимо преобразовать эти данные в формат, пригодный для передачи по сети. Этот процесс называется сериализацией (также маршалингом или упаковкой) – преобразование структуры данных или состояния объекта в формат, который может быть сохранен (например, в файле или буфере памяти) или передан и восстановлен позже. Обратный процесс – восстановление исходной структуры данных из сериализованного формата – называется десериализацией (анмаршалингом или распаковкой).

Выбор формата сериализации данных является важным архитектурным решением, влияющим на производительность, объем передаваемого трафика, сложность разработки и отладки. Рассмотрим наиболее распространенные форматы.

1.2.1 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) является легковесным текстовым форматом обмена данными, основанным на подмножестве синтаксиса языка JavaScript. Он был популяризирован Дугласом Крокфордом в начале 2000-х годов и быстро стал стандартом де-факто для многих веб-сервисов и API благодаря своей простоте и удобочитаемости как для человека, так и для машины.

Структура JSON основана на двух основных конструкциях:

— Объект (Object): Неупорядоченное множество пар "ключ/значение". Ключ – это строка, а значение может быть строкой, числом, булевым значением (true/false), массивом, другим объектом или специальным значением null. Объекты заключаются в фигурные скобки {}.

— Массив (Array): Упорядоченная коллекция значений. Значения могут быть любого типа, допустимого в JSON. Массивы заключаются в квадратные скобки [].

Примитивные типы данных в JSON включают строки (в двойных кавычках), числа (целые или с плавающей точкой), булевы значения (true, false) и null.

Преимущества JSON:

— Читаемость: JSON легко читается и понимается человеком, что упрощает разработку и отладку.

— Простота: Синтаксис JSON прост и интуитивно понятен.
— Широкая поддержка: Большинство языков программирования имеют встроенные или сторонние библиотеки для парсинга и генерации JSON.
— Нативность для JavaScript: Поскольку JSON является подмножеством JavaScript, его обработка в JavaScript-окружениях (браузеры, Node.js) очень эффективна с помощью встроенных методов JSON.parse() и JSON.stringify().
— Независимость от платформы и языка: Хотя JSON и основан на JavaScript, он является языково-независимым форматом данных.

Недостатки JSON:

— Избыточность: Будучи текстовым форматом, JSON часто содержит избыточную информацию. Ключи объектов передаются как строки в каждом сообщении, что увеличивает общий объем данных, особенно для массивов однотипных объектов с повторяющимися ключами. Пробелы и символы форматирования также добавляют к размеру, хотя они могут быть удалены при минимизации.
— Отсутствие строгой типизации и схем: JSON сам по себе не определяет схему данных. Типы данных ограничены (нет специального типа для дат, двоичных данных и т.д. – даты обычно передаются как строки ISO 8601 или числа timestamp). Отсутствие схем может приводить к ошибкам при несоответствии ожиданиям клиента или сервера.
— Производительность при больших объемах: Парсинг больших JSON-структур может быть ресурсоемким по сравнению с некоторыми бинарными форматами.
— Отсутствие поддержки комментариев: Стандарт JSON не допускает комментариев в данных, что иногда неудобно при ручном редактировании конфигурационных файлов в формате JSON.

Несмотря на недостатки, JSON остается чрезвычайно популярным благодаря своей простоте и удобству в экосистеме веб-разработки. Пример JSON-объекта:

```
{
        "id": 1,
        "name": "Аспирин",
        "quantity": 20,
        "unit": "таб.",
        "active": true,
```

```
7               "notes": null
8       }
```

1.2.2 XML (Extensible Markup Language)

XML (Extensible Markup Language) – это текстовый формат, разработанный консорциумом W3C для хранения и передачи структурированных данных. XML был разработан как более гибкий и расширяемый преемник HTML, предназначенный для описания данных, а не их представления.

Ключевые особенности XML:

- Теговая структура: Данные представляются в виде дерева элементов, где каждый элемент обрамлен открывающим и закрывающим тегами (например, <medicine>...</medicine>). Элементы могут иметь атрибуты.
- Самоописываемость: Имена тегов могут быть произвольными и описывать семантику данных.
- Поддержка схем: XML позволяет определять строгие схемы данных с помощью XSD (XML Schema Definition) или DTD (Document Type Definition), что обеспечивает валидацию структуры и типов данных.
- Поддержка пространств имен: Позволяет избегать конфликтов имен при комбинировании XML-документов из разных источников.

Пример XML-документа, эквивалентного JSON-примеру выше:

```
1       <medicine id="1">
2       <name>Аспирин</name>
3       <quantity>20</quantity>
4       <unit>таб.</unit>
5       <active>true</active>
6       <notes/>
7       </medicine>
```

Сравнение с JSON:

- Многословность: XML значительно более многословен, чем JSON, из-за необходимости использования открывающих и закрывающих тегов для каждого элемента данных. Это приводит к большему объему передаваемых данных.
- Сложность парсинга: Парсеры XML обычно сложнее и медленнее, чем парсеры JSON.
- Строгость и валидация: Благодаря схемам (XSD), XML обеспечивает

более строгую типизацию и валидацию данных по сравнению с JSON "из коробки".

— Читаемость: Хотя XML и читаем человеком, его структура с большим количеством тегов может восприниматься как менее удобная по сравнению с компактной нотацией JSON.

В современных веб-API XML используется значительно реже, чем JSON, особенно для легковесных сервисов и мобильных приложений, уступив место последнему из-за его простоты и меньшей избыточности. Однако XML остается важным стандартом в корпоративных системах, для обмена документами и в областях, где важна строгая валидация по схемам (например, SOAP-сервисы).

1.2.3 Бинарные форматы сериализации

В отличие от текстовых форматов, таких как JSON и XML, бинарные форматы представляют данные непосредственно в виде последовательности байт, что позволяет достичь значительно большей компактности и, зачастую, более высокой скорости сериализации и десериализации. Эти преимущества особенно важны для высоконагруженных систем, приложений с ограниченной пропускной способностью сети (например, мобильных приложений или IoT-устройств) и при передаче больших объемов данных.

Общие принципы бинарных форматов:

— Компактное представление типов: Числа, булевы значения и другие примитивные типы кодируются с использованием минимально необходимого количества байт, а не их текстового представления.

— Эффективное кодирование строк: Строки часто предваряются их длиной.

— Оптимизация ключей: Имена полей (ключи объектов) могут заменяться числовыми идентификаторами или индексами, что существенно сокращает избыточность по сравнению с JSON, где ключи повторяются в каждой записи.

— Поддержка схем (часто): Многие бинарные форматы (например, Protocol Buffers, Avro) требуют определения схемы данных, что обеспечивает строгую типизацию и облегчает обратную совместимость при изменении структуры данных.

Недостатки бинарных форматов:

— Нечитаемость человеком: Бинарные данные не предназначены для прямого чтения или редактирования человеком без специальных инструментов.
— Сложность отладки: Отладка проблем с бинарными данными может быть сложнее, чем с текстовыми.
— Зависимость от библиотек и схем: Для работы с бинарными форматами требуются соответствующие библиотеки сериализации/десериализации. Если формат основан на схемах, то и клиент, и сервер должны иметь доступ к актуальной схеме.

Рассмотрим один из популярных бинарных форматов, который будет использоваться для сравнения в данной работе.

MessagePack – это эффективный формат бинарной сериализации, который позиционируется как "JSON, только быстрый и маленький" [?]. Он стремится быть таким же простым в использовании, как JSON, но предлагать лучшую производительность и меньший размер данных.

Ключевые особенности MessagePack:

— Компактность: Целые числа кодируются с использованием переменного количества байт (например, маленькие числа занимают один байт). Строки предваряются своей длиной. Карты (объекты) и массивы также имеют компактное представление.
— Скорость: Благодаря бинарному представлению и отсутствию необходимости парсить текстовые строки для ключей и значений (кроме самих строк), (де)сериализация MessagePack обычно быстрее, чем JSON.
— Поддержка типов данных: MessagePack поддерживает все типы данных JSON (null, boolean, number, string, array, map/object), а также добавляет поддержку бинарных данных (byte arrays) и расширяемые типы (ext types), которые позволяют определять пользовательские типы данных, включая эффективное представление для timestamp.
— Отсутствие схем по умолчанию: Как и JSON, MessagePack по умолчанию не требует определения схемы, что упрощает его использование, но может приводить к тем же проблемам с типизацией, что и у JSON, если не используются внешние механизмы валидации.

Сравнение с JSON по размеру часто показывает, что MessagePack может уменьшить объем данных на 30-50% и более, в зависимости от структуры

данных. Например, объект из листинга ?? в MessagePack будет представлен более компактной последовательностью байт, где ключи "id "name"и т.д. будут закодированы как строки, но сами значения (числа, булевы) будут занимать меньше места. Если бы ключи были предопределены и заменены на числовые идентификаторы (что возможно с MessagePack через кастомные решения или библиотеки, работающие поверх него), экономия была бы еще больше.

Существует множество реализаций библиотек для работы с MessagePack на различных языках программирования, включая JavaScript (для Node.js и браузеров).