**КТ № 8. Сервер. Получение данных от клиента. Шаблоны.**

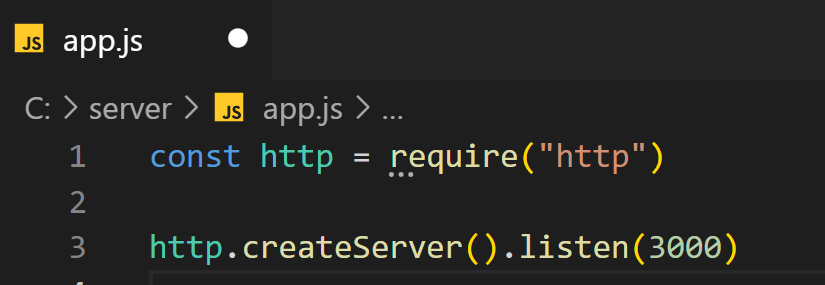
**Время на выполнение практической работы:** 4 часа.

**Теоретические сведения к выполнению практической работы:**

**Создание сервера.**

Для работы с сервером и протоколом http в Node.js используется модуль http.

Чтобы создать сервер, следует вызвать метод http.createServer():



Метод createServer() возвращает объект http.Server. Но чтобы сервер мог прослушивать и обрабатывать входящие подключения, у объекта сервера необходимо вызвать метод listen(), в который в качестве параметра передается номер порта, по которому запускается сервер.

Для обработки подключений в метод createServer можно передать специальную функцию:



Эта функция принимает два параметра:

request: хранит информацию о запросе

response: управляет отправкой ответа

**Request**

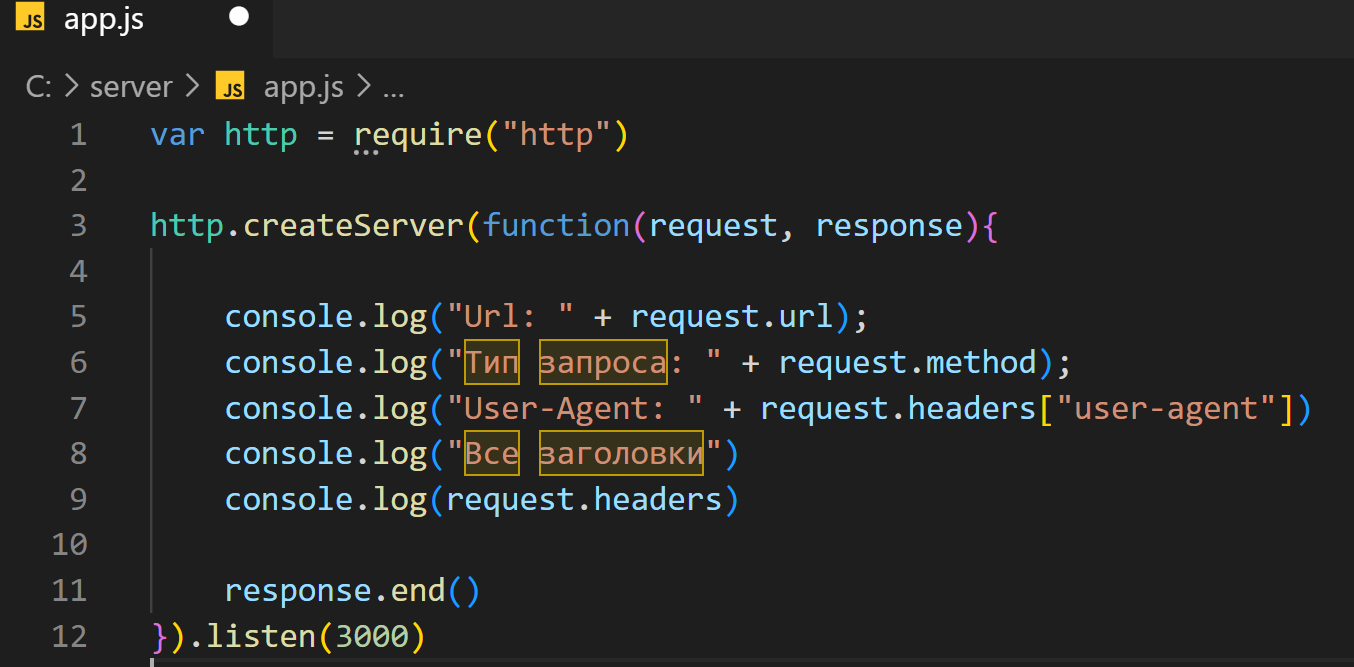
Параметр request позволяет получить информацию о запросе и представляет объект http.IncomingMessage. Отметим некоторые основные свойства этого объекта:

headers: возвращает заголовки запроса

method: тип запроса (GET, POST, DELETE, PUT)

url: представляет запрошенный адрес

Например, определим следующий файл app.js:



Запустим его и обратимся в браузере по адресу <http://localhost:3000/index.html>:



**Response**

Параметр response управляет отправкой ответа и представляет объект http.ServerResponse. Среди его функциональности можно выделить следующие методы:

statusCode: устанавливает статусный код ответа

statusMessage: устанавливает сообщение, отправляемое вместе со статусным кодом

setHeader(name, value): добавляет в ответ один заголовок

write: пишет в поток ответа некоторое содержимое

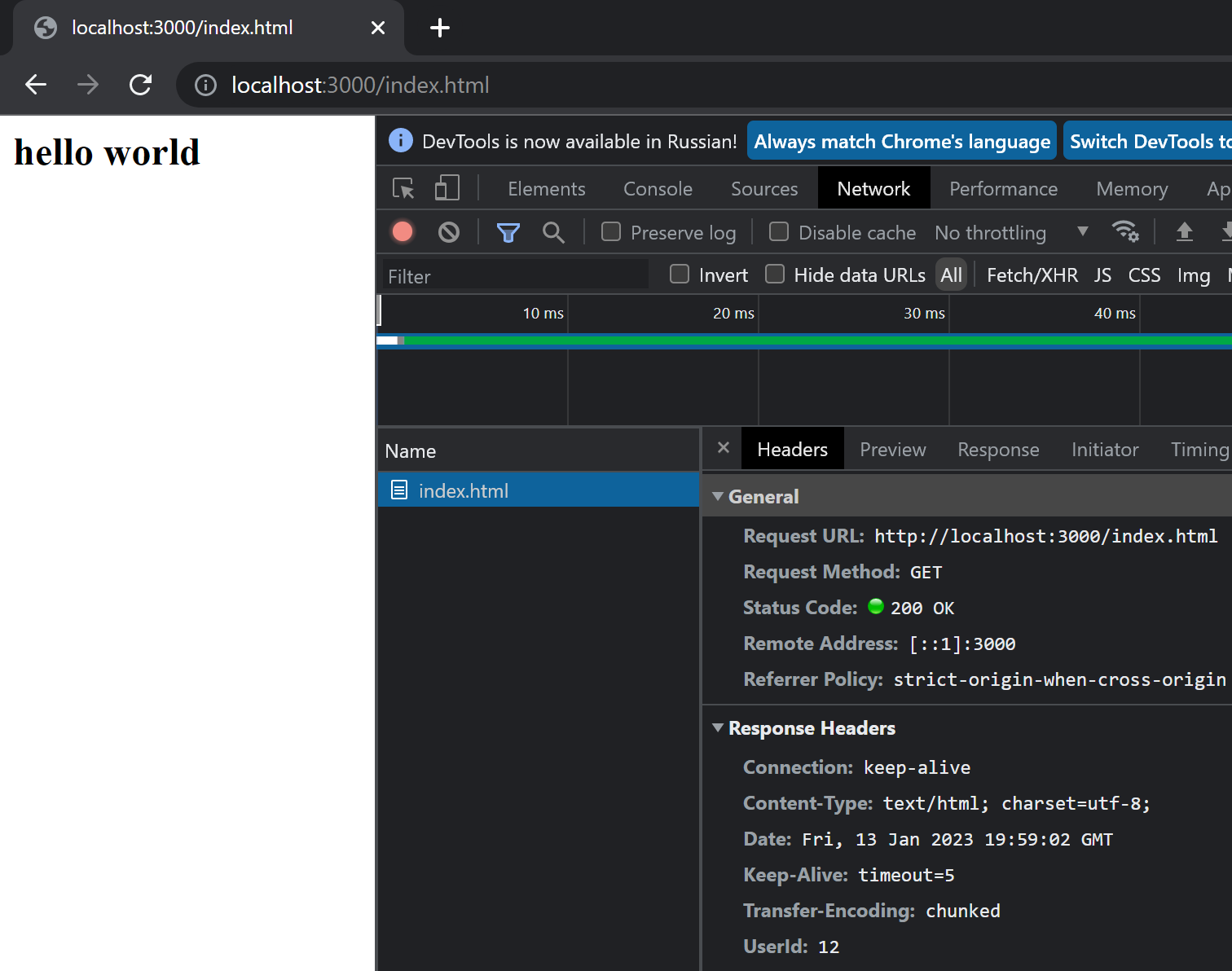
writeHead: добавляет в ответ статусный код и набор заголовков

end: сигнализирует серверу, что заголовки и тело ответа установлены, в итоге ответ отсылается клиента. Данный метод должен вызываться в каждом запросе.

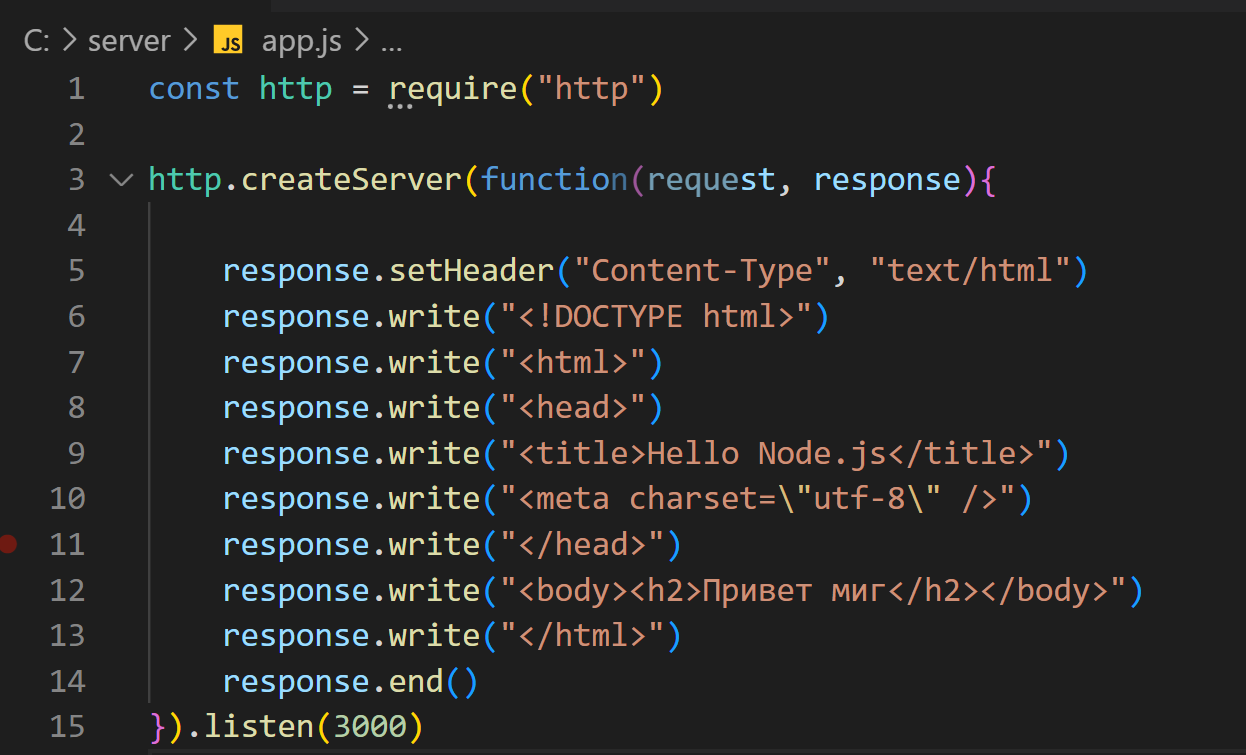
Например, изменим файл app.js следующим образом:

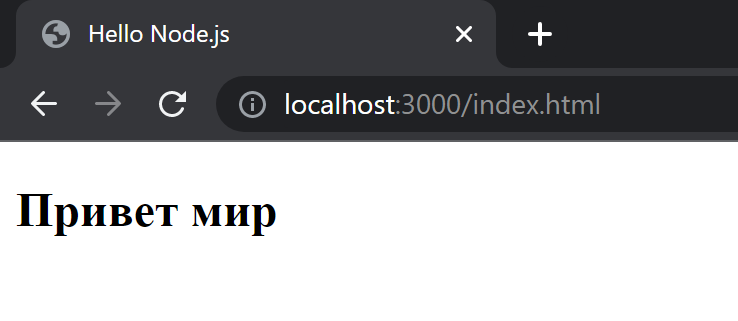


Запустим файл и обратимся в браузере к приложению:



Если предстоит отправить довольно большой ответ, то мы можем несколько раз вызвать метод write(), последовательно оправляя в исходящий поток каждый кусочек информации. Например, отправим код более менее полноценной веб-страницы:



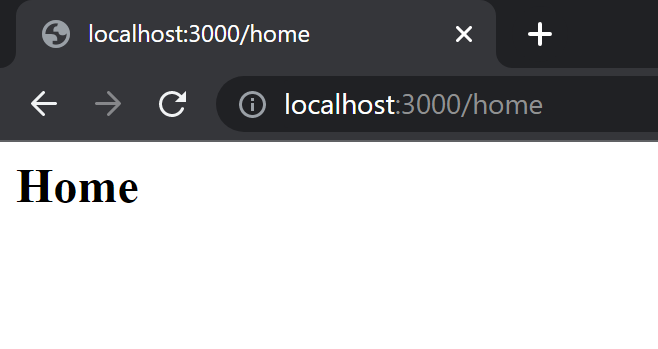


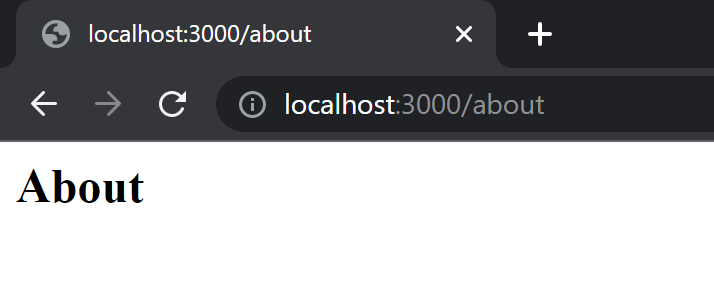
**Маршрутизация**

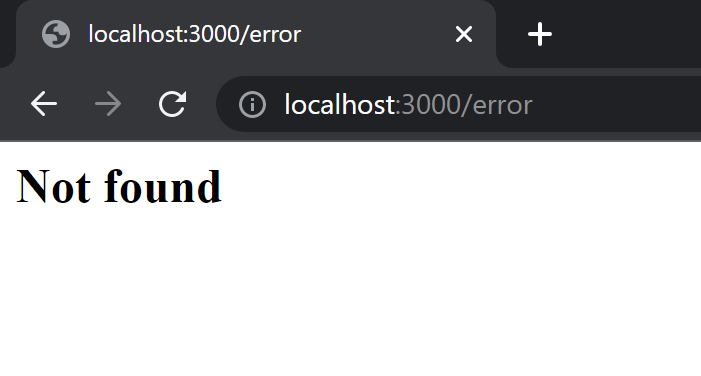
По умолчанию Node.js не имеет встроенной системы маршрутизации. Обычно она реализуется с помощью специальных фреймворках типа Express, о котором речь пойдет в следующей главе. Однако если необходимо разграничить простейшую обработку пары-тройки маршрутов, то вполне можно использовать для этого свойство url объекта Request. Например:



В данном случае обрабатываются три маршрута. Если идет обращение к корню сайта или по адресу localhost:3000/home, то пользователю выводится строка "Home". Ели обращение идет по адресу localhost:3000/about, то пользователю в браузере отображается строка About и так далее. Если запрошенный адрес не соответствует ни одному маршруту, то выводится заговлок "Not Found".



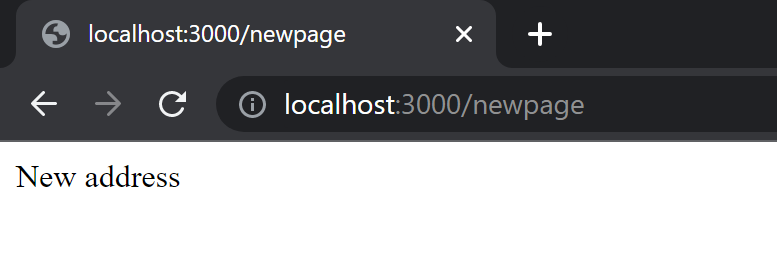




**Переадресация**

Переадресация предполагает отправку статусного кода 301 (постоянная переадресация) или 302 (временная переадресация) и заголовка Location, который указывает на новый адрес. Например, выполним переадресацию с адреса localhost:3000/ на адрес localhost:3000/newpage





**Отправка файлов**

Отправка статических файлов - довольно частая задача в построении и функционировании веб-приложения. Рассмотрим, как отправлять файлы в приложении на Node.js.

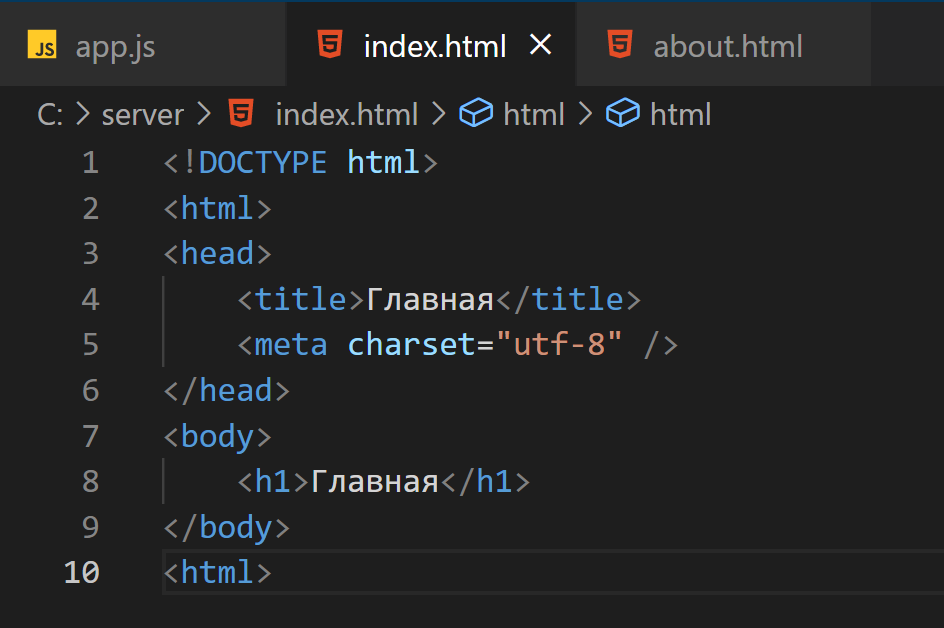
Пусть в каталоге проекта у нас будут три файла:

app.js

about.html

index.html

Наряду с файлом приложения app.js определим два html-файла. В файле index.html определим следующий код:



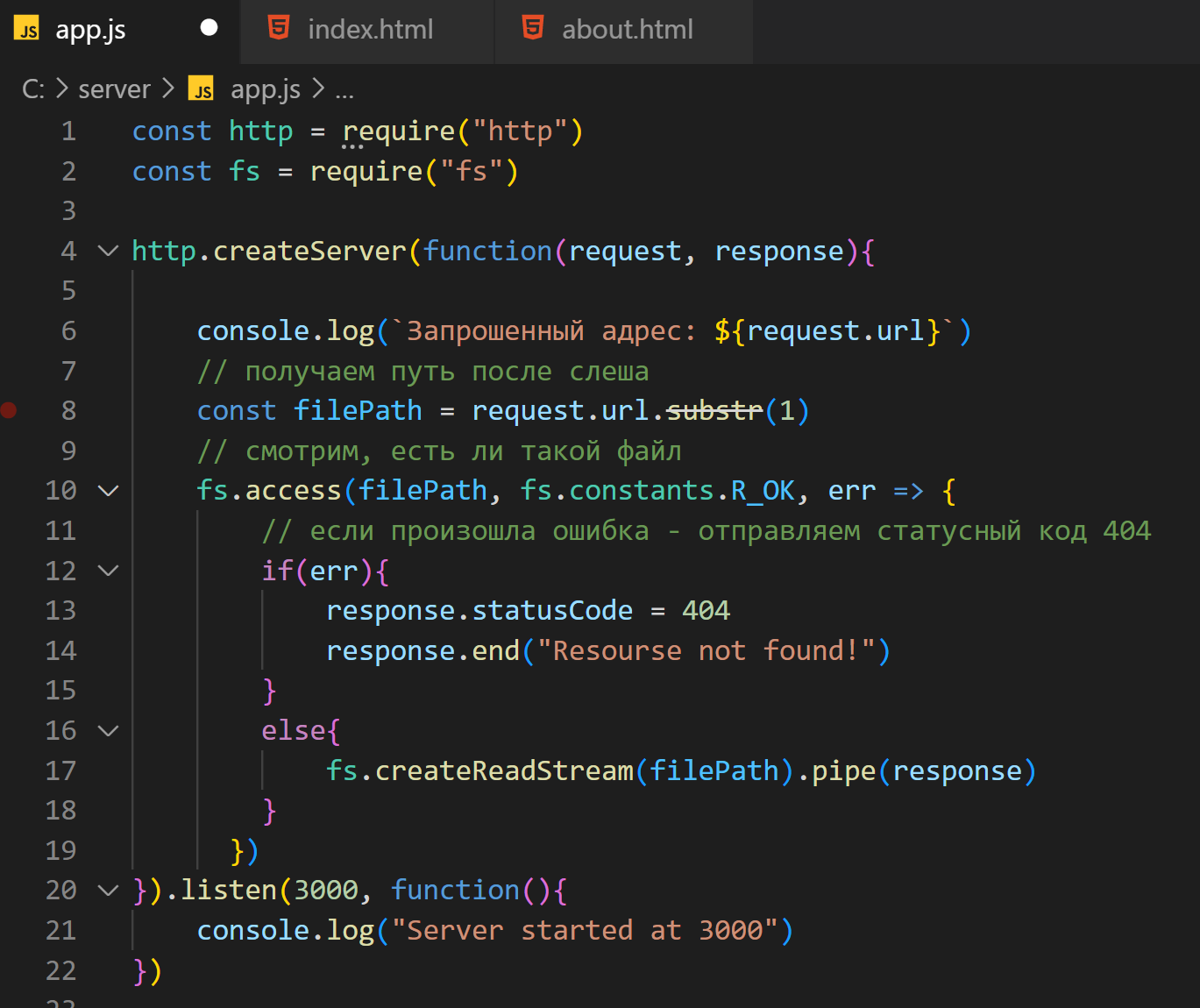
Аналогично определим код в файле about.html:



Наша задача будет заключаться в том, чтобы отправить их содержимое пользователю.

**Первый способ**

Для считывания файла может применяться метод fs.createReadStream(), который считывает файл в поток, и затем с помощью метода pipe() мы можем связать считанные файлы с потоком записи, то есть объектом response. Итак, поместим в файл app.js следующий код:

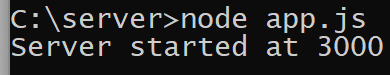


Вначале мы получаем запрошенный адрес. Допустим, запрошенный адрес будет соответствовать напрямую пути к файлу на сервере. Затем с помощью асинхронной функции fs.access проверяем доступность файла для чтения. Первый параметр функции - путь к файлу. Второй параметр - опция, относительно которой проверяется доступ. В данном случае значение fs.constants.R\_OK говорит о том, что мы проверяем права на чтение из файла. Третий параметр функции - функция обратного вызова, которая получает объект ошибки. Если произошла ошибка (файл не доступен для чтения или вовсе не найден), посылаем статусный код 404.

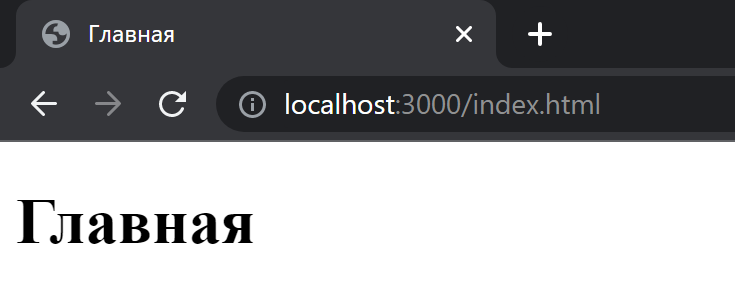
Для отправки файла применяется цепочка методов



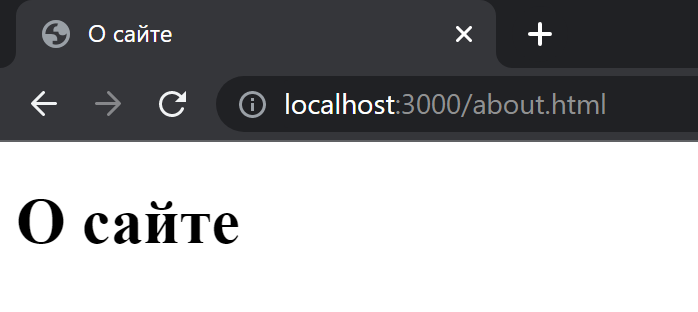
Метод fs.createReadStream("some.doc") создает поток для чтения - объект fs.ReadStream. Для получения данных из потока вызывается метод pipe(), в который передается объект интерфейса stream.Writable или поток для записи. А именно таким и является объект http.ServerResponse, который реализует этот интерфейс.



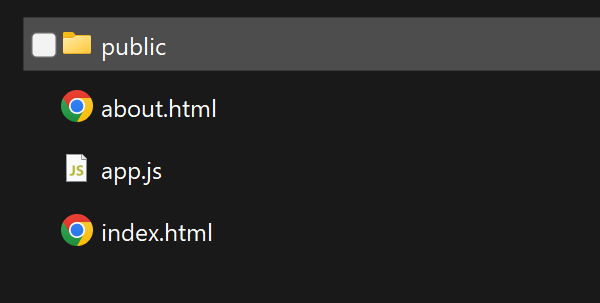
Запустим приложение и в браузере обратимся по адресу "http://localhost:3000/index.html":



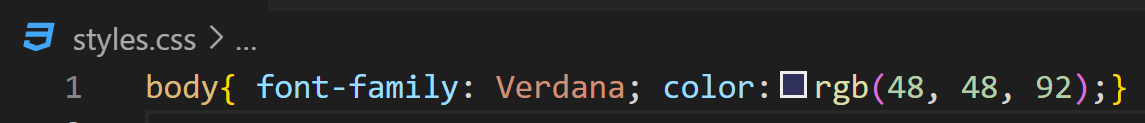
Аналогично мы можем обратиться по адресу "http://localhost:3000/about.html":



В данном случае отправляются файлы html, но подобным образом мы можем отправлять самые разные файлы. Например, определим в проекте папку public.



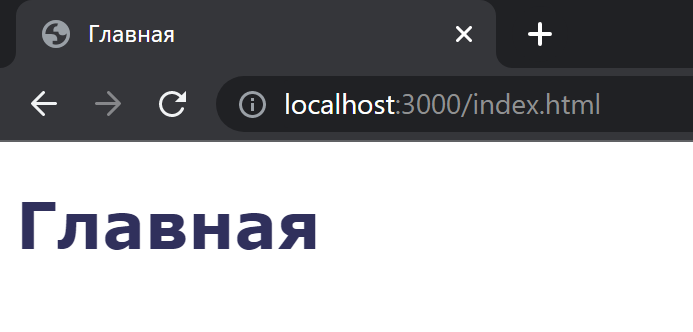
И в ней создадим новый файл styles.css со следующим содержимым:



Применим эти стили на станице index.html:

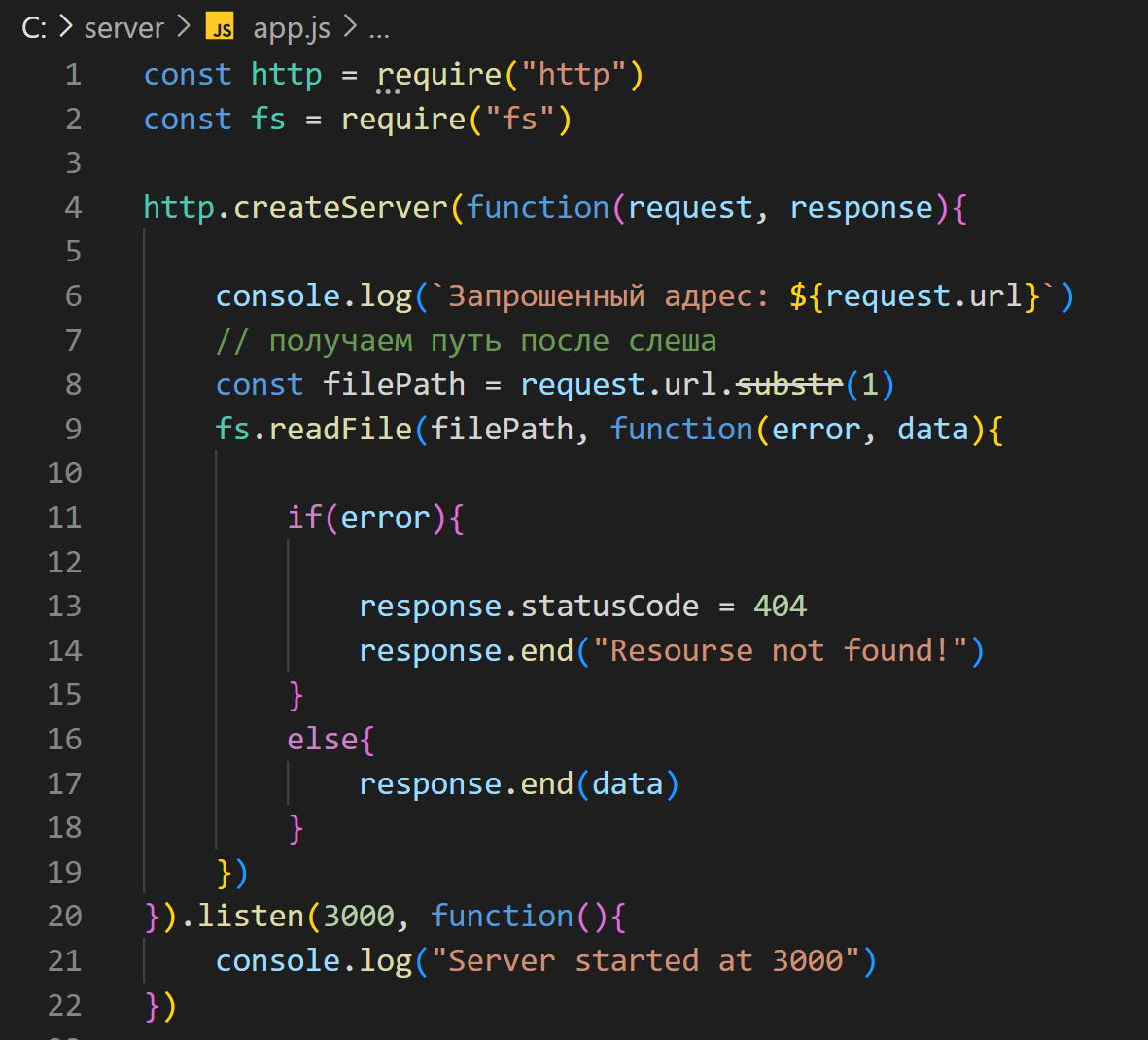


И затем обратимся к index.html:



**Второй способ**

Второй способ представляет чтение данных с помощью функции fs.readFile() и отправка с помощью метода response.end():



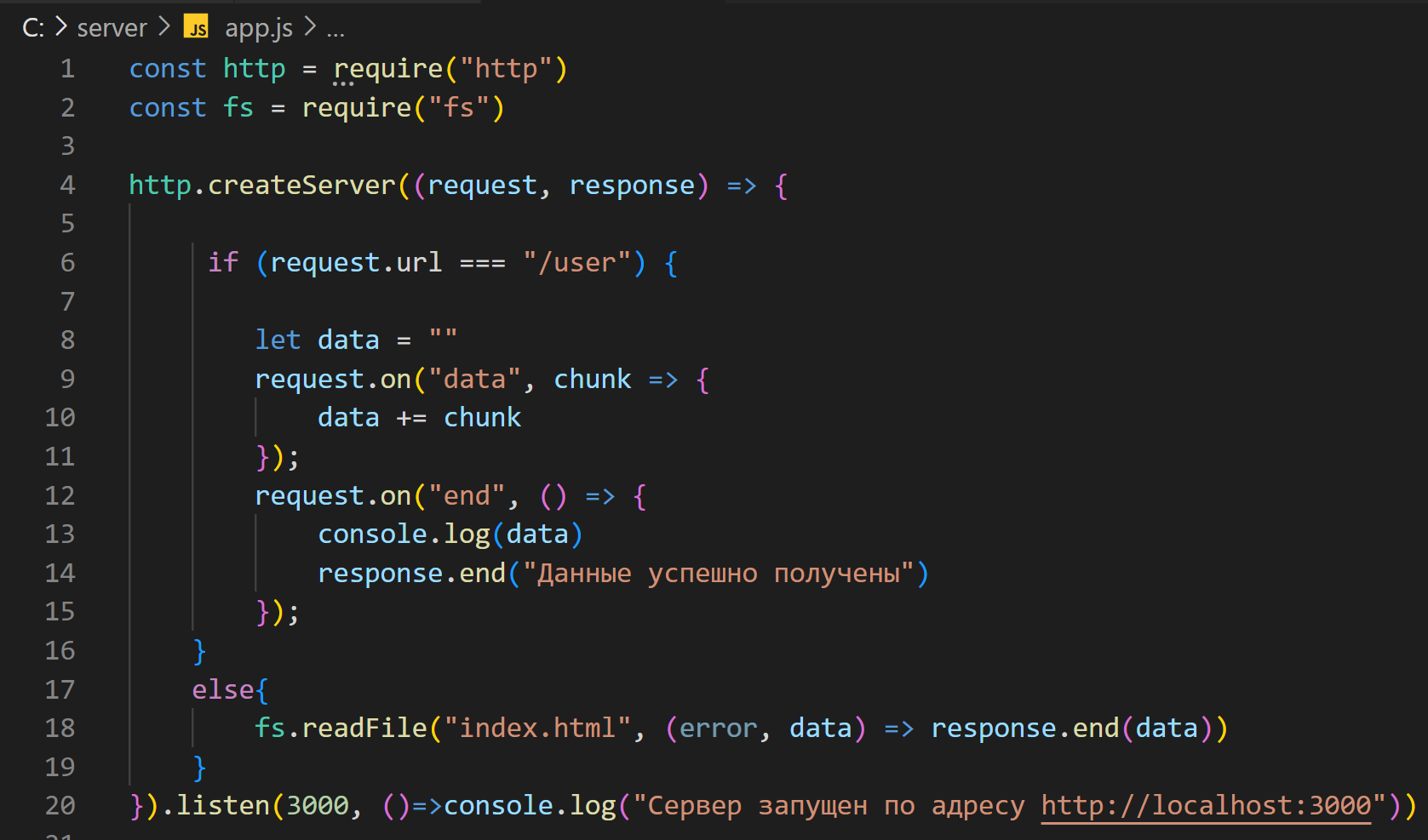
**Получение данных от клиента**

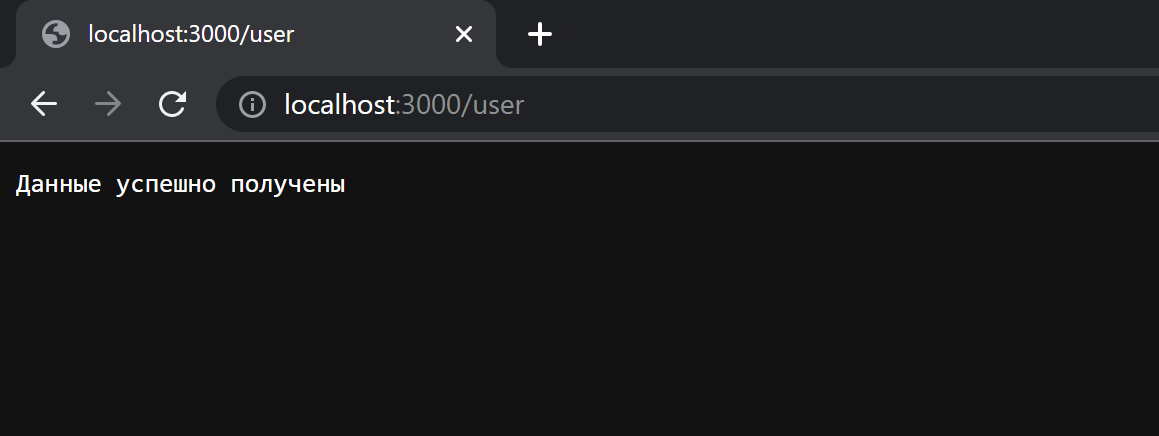
Все данные запроса передаются через объект request - первый параметр функции обратного вызова Функция, которая передается в функцию http.createServer(). Однако получить из него отправленные клиентом данные может представлять сложность. Прежде всего потому, что функция обратного вызова в http.createServer() вызывается при получении всех HTTP-заголовков, но не тела запроса. Кроме того, сам объект request представляет поток, который получает данные отдельными чанками. Соответственно для получения данных из запроса нам необходимо прослушивать этот поток и последовательно извлекать их него полученные чанки информации.

В этом случае для получения данных мы можем использовать разные способы.

**Обработка событий data и end объекта request**

Первый способ представляет прослушивание события data обработчик которого получает пришедшие чанки информации. А когда все данные получены, срабатывает событие end. Например, определим следующий простейший сервер:



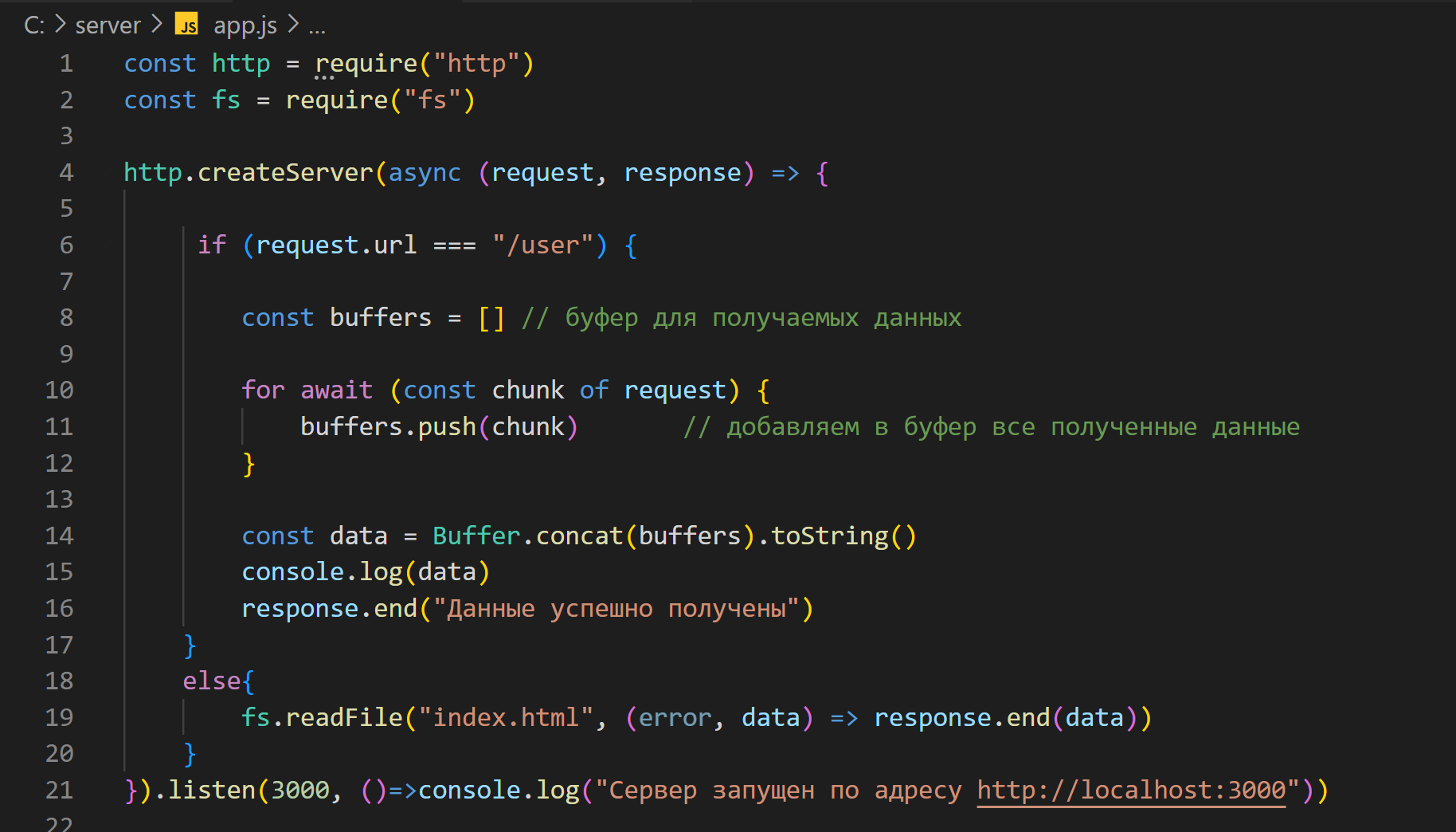


При обращении по адресу "/user" с помощью метода request.on устанавливаем для события data функцию-обработчик. Эта функция в качестве параметра получает очередной пришедший кусочек оправленных данных в виде параметра chunk. В данном случае мы ожидаем, что клиент будет отправлять текст. И в функции-обработчике добавляем полученные данные к переменной data. И каждый раз, когда приложение получит очередную порцию данных, будет срабатывать данные обработчик и добавлять данные к переменной data.

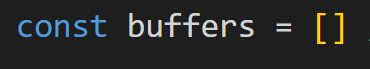
Когда все данные будут получены, сработает событие end, в функции-обработчике которого мы уже будем знать, что все данные получены и мы можем производить с ними необходимые нам манипуляции. В данном случае полученные данные просто выводятся на консоль, а в ответ клиенту отправляется некоторое уведомление.

**Перебор асинхронного итератора request**

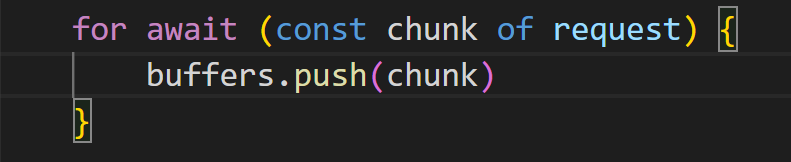
Второй способ представляет перебор объекта request, который также предоставляет асинхронный итератор для получаемых данных:



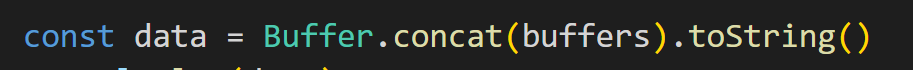
Здесь вначале определяем буфер для добавляемых данных:



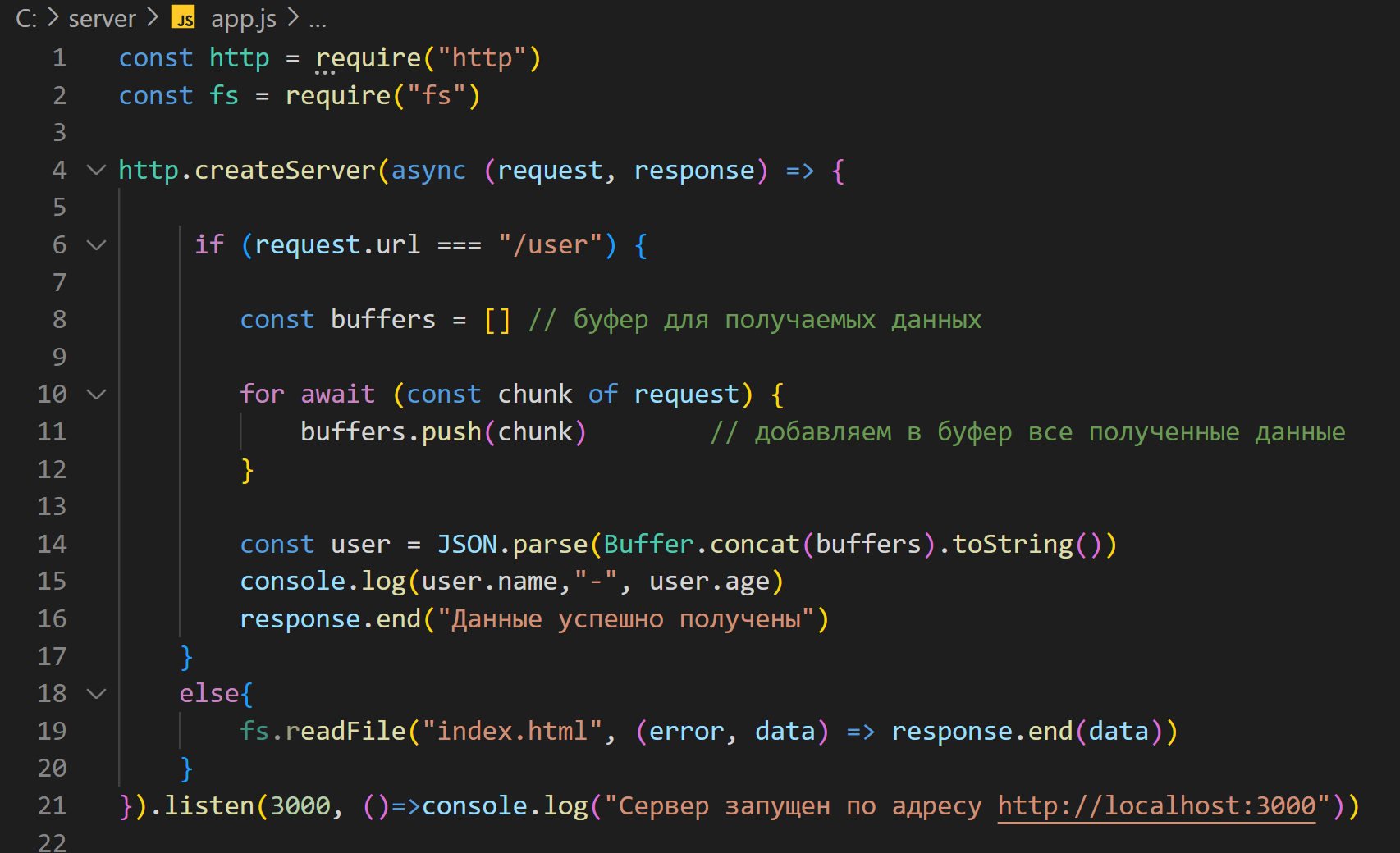
Далее перебираем объект request, получая из него данные и добавляя их в буфер buffers:



И в конце с помощью функции Buffer.concat() объединяем все полученные данные и преобразуем их в строку методом toString():



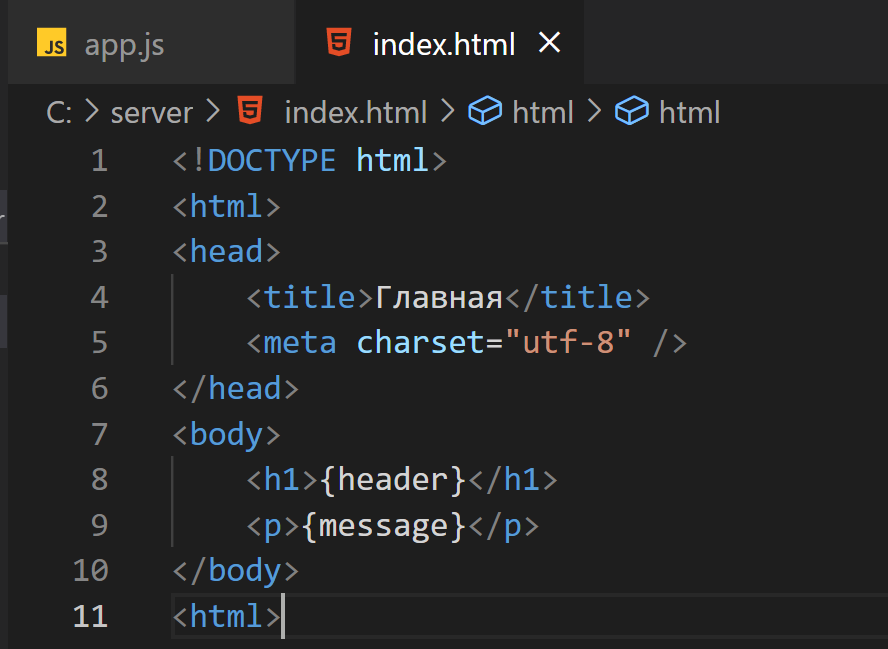
При этом мы не ограничены получением только текстовых данных. Например, получим данные в формате json:



Здесь предполагается, что клиент посылает серверу объект в формате JSON, у которого есть свойства name и age. И с помощью функции JSON.parse() мы можем распарсить полученные данные в формат json.

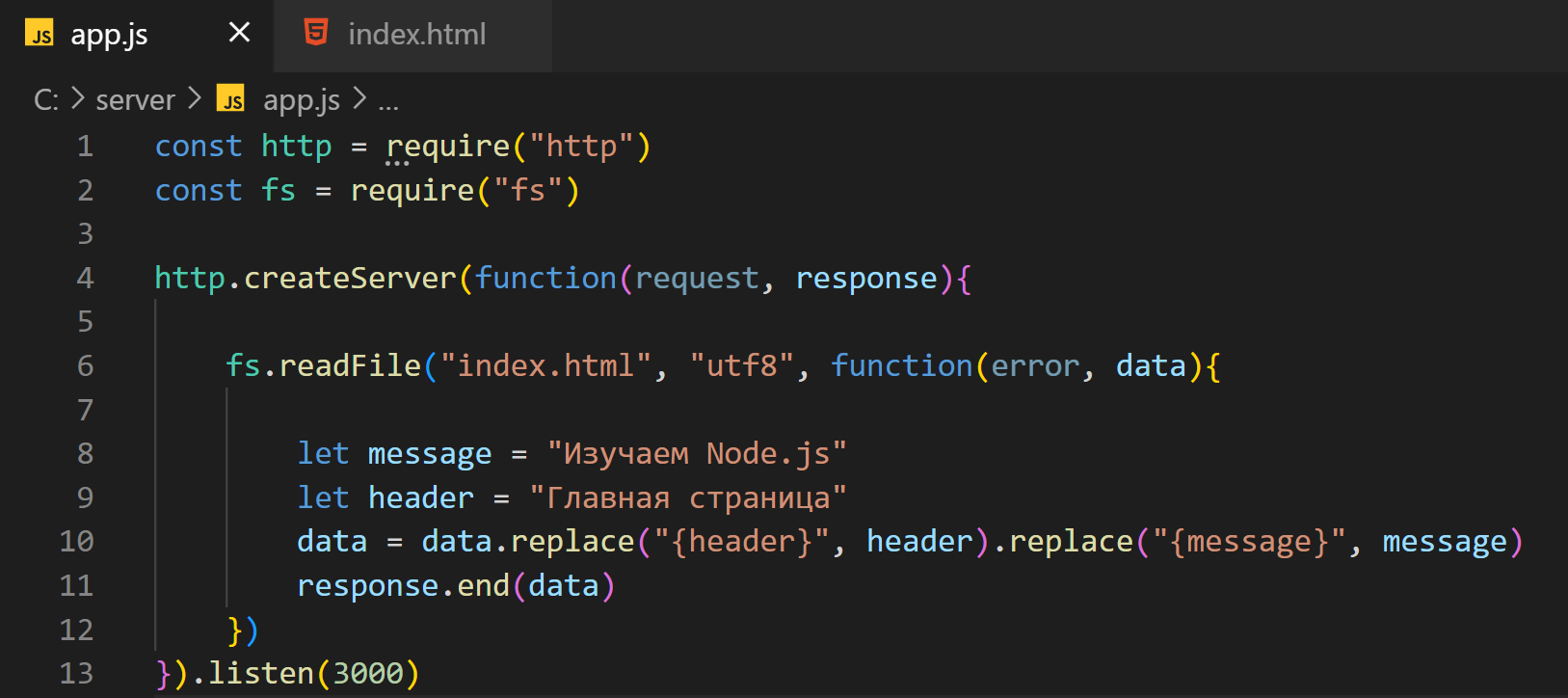
**Шаблоны**

Выше использовался файл html, который имел статичное неизменяющееся содержимое. Однако мы также можем применять специальные инструменты - шаблоны, вместо которых в файл будет вставляться какой-то определенный текст. Например, изменим файл index.html следующим образом:



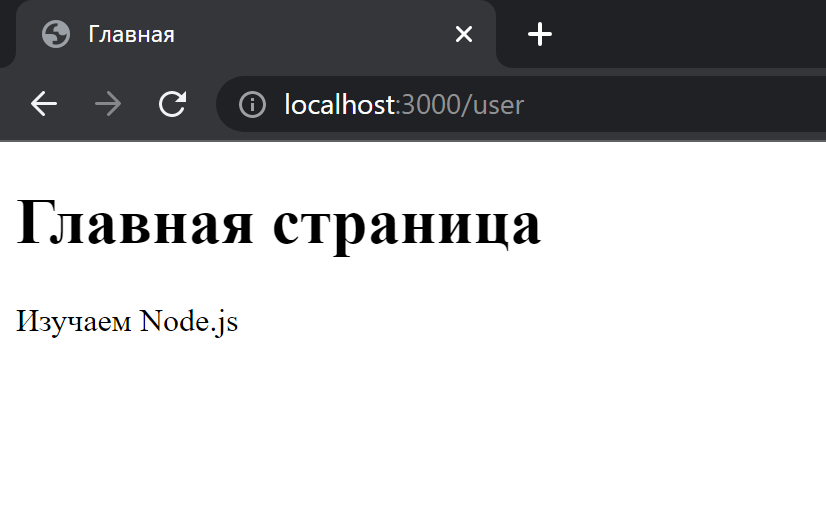
Вместо конкретного содержимого здесь определены плейсхолдеры "{header}" и "{message}", вместо которых может вставляться любой текст.

Изменим файл app.js:



Здесь получаем содержимое файла и проводим его дополнительную обработку, заменяя плейсхолдеры на конкретный текст с помощью метода data.replace().

При обращении к приложению мы получим полноценную html-страницу без плейсхолдеров:



**Этапы выполнения:**

1. ***Создайте сервер. Воспользуйтесь параметрами request и response. Запустите файл, подтвердите скриншотом изменение UserId;***
2. ***Поработайте с маршрутизацией. Обработайте четыре маршрута: home, about, contact и обработка адреса, не соответствующего ни одному из маршрутов;***
3. ***Результат зафиксировать при помощи браузера, сделав скриншот каждой из четырех страниц;***
4. ***Реализовать переадресацию. Результат зафиксировать при помощи браузера;***
5. ***Отправьте файл двумя способами;***
6. ***Получите данные от клиента двумя способами: при помощи прослушивания события data и при помощи перебора объекта request;***
7. ***Используйте шаблоны.***