Building a To-Do List Server

Consider the code in **todo-server.js**. Add functionality to this file so the server can respond with the to-do list HTML/JavaScript.

We may want to share the list data among multiple clients. In this case, the server will act as a centralized store of the information.

Clients can request the list data (i.e., with a GET request) or request to change the list data (i.e., with a POST or PUT request)

todo-server.js

```
const http = require('http');
const fs = require('fs');

const server = http.createServer(function (request, response) {
    if (request.method === 'GET'){
        if (request.url === "/todo.html" || request.url === "/"){
            fs.readFile("todo.html", function(err,data){
                if (err){
                    response.statusCode = 500;
                    response.end("Error reading file.");
                }else{
                    //we have a data at this point
                    response.statusCode = 200;
                    response.end(data);
                }
            });
        }
    }else{
        response.statusCode = 404;
        response.end("Unknown resource");
    }
});
//Server listens on port 3000
server.listen(3000);
console.log('Server running at http://127.0.0.1:3000/');
```

```
else{
    response.statusCode = 404;
    response.end("Unknown resource");
}
```

We should also set headers:

```
response.setHeader(
"Content-Type",
 "text/html");
```

todo-server.js

```
const http = require('http');
const fs = require("fs");
const server = http.createServer(function (request, response) {
    console.log(request.url);
    if(request.method === "GET"){
        if(request.url === "/" || request.url === "/todo.html"){
            //read the todo.html file and send it back
            fs.readFile("todo.html", function(err, data){
                if(err){
                    response.statusCode = 500;
                    response.write("Server error.");
                    response.end();
                    return;
                }
                response.statusCode = 200;
                response.setHeader("Content-Type", "text/html");
                response.write(data);
                response.end();
            });
        }else if(request.url === "/todo.js"){
            //read todo.js file and send it back
            fs.readFile("todo.js", function(err, data){
                if(err){
                    response.statusCode = 500;
                    response.write("Server error.");
                    response.end();
                    return;
                }
                response.statusCode = 200;
                response.setHeader("Content-Type", "application/javascript");
                response.write(data);
                response.end();
            });
```

```
}else if(request.url === "/listdata"){
        //respond with JSON of list object
```

```
        //Add any more 'routes' and their handling code here
        //e.g., GET requests for "/list", POST request to "/list"
        }else{
            response.statusCode = 404;
            response.write("Unknwn resource.");
            response.end();
        }
    }else if(request.method === "POST"){
        //any handling in here
        response.statusCode = 404;
        response.write("Unknwn resource.");
        response.end();
    }
});

//Server listens on port 3000
server.listen(3000);
console.log('Server running at http://127.0.0.1:3000/');
```

Building a To-Do List Server

This involves at least three main steps on the server:

1. Create variable to store the list data on the server
2. Add route handler for GET requests to list URL (e.g., /list)
3. Add route handler for POST requests to list URL (e.g., /list)

Remember: you can JSON.stringify(obj) any object to send in response

Building a To-Do List Server

The client will also require some changes:

1. When new items are added to the list, use a POST request to send that new item to the server
2. Intermittently (e.g., every X seconds) make a GET request for the list data and update the page contents

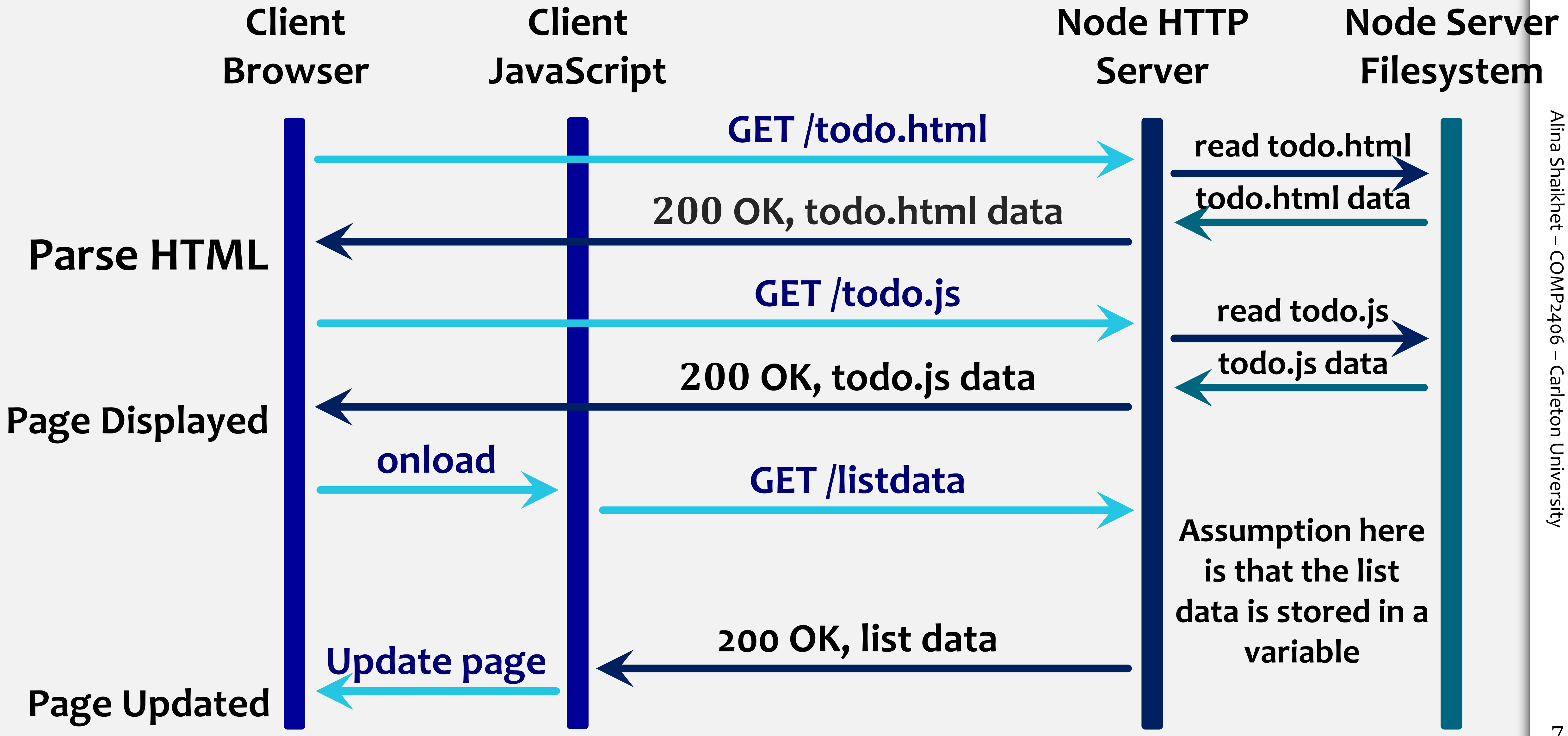
These steps are facilitated by the `XMLHttpRequest`.

Building a To-Do List Server

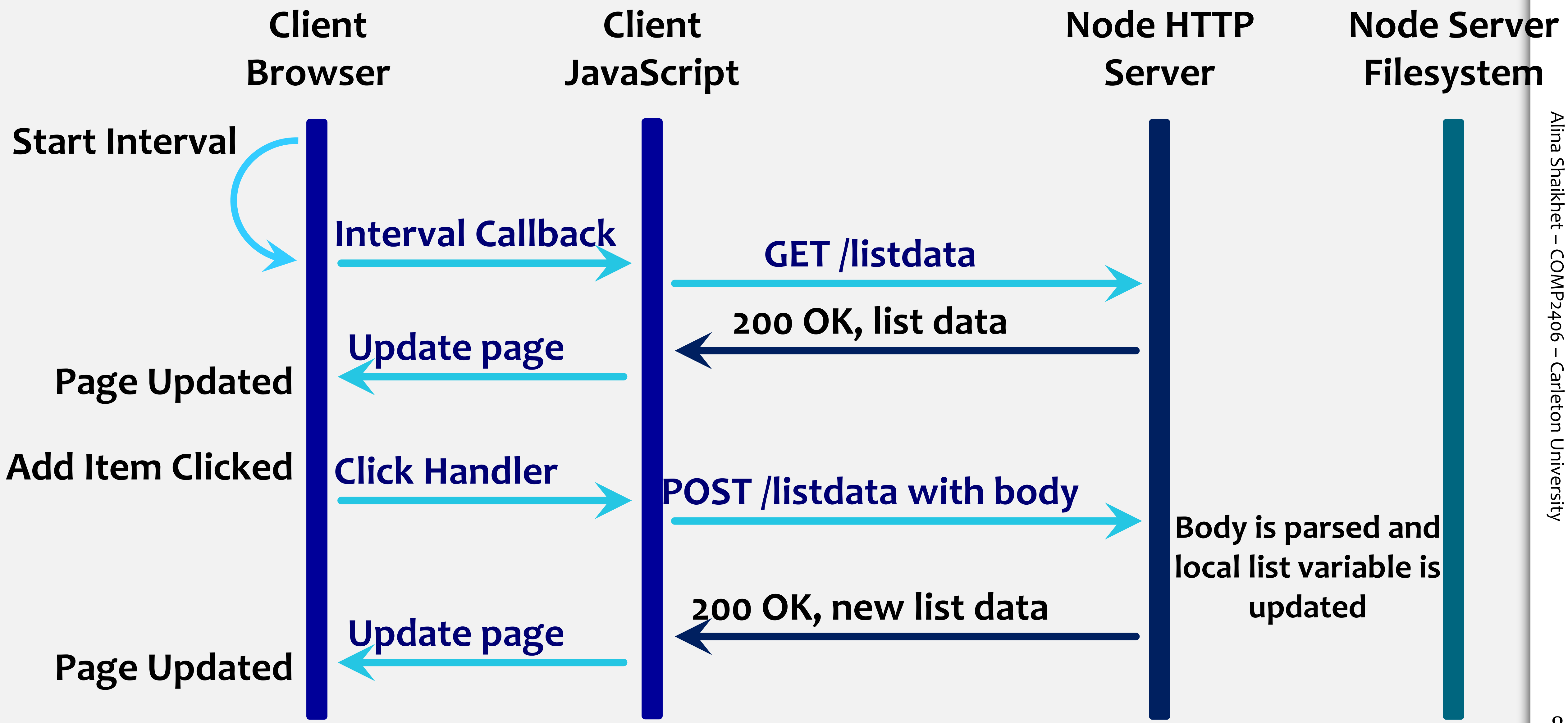
If you have a good design, the client changes should be minimal

For example, if you are rendering the page contents from a single object, implement your server so it sends the same object structure to the client

The To-Do List Interactions



The To-Do List Interactions



Building a To-Do List Server

An important question: how will you indicate what operation you want to perform on the server

e.g., add an item, remove items, etc.

Different routes/URLs?

Different HTTP methods?

Summary

We have a way of accepting requests, handling them, and sending responses. The ability to do this just comes down to organization of that data.

Give meaning to all those URLs.

We can add as much logic as we need into the handler function to build a complex web system

This would get messy quickly and involve a lot of manual work on our part

Throughout the course, we will look at some ways to build these systems in a more efficient, scalable, and extensible way

But it is good to understand what is happening in the basic sense before we get into those details