**What is server-side programming?**

Before getting into the specifics of building web apps with Node, we need to get a good picture of what server-side programming is all about *in general*, regardless of programming language.

**Web apps expose resources**

A server-side web app is simply a program whose purpose is to expose resources (data, files, etc.) to other applications. A single server supports many clients. A web server's clients can be web browsers, mobile apps, or other servers.

Client applications *request* resources from a server (for instance, an index.html page, data representing a set of customers, a style sheet).

Clients can also send data to the server (for instance, the email address and password submitted by a user in a sign up form) so it can be stored or otherwise processed.

The types of server-side apps we'll look at in this course expose two kinds of resources. First, they can *send files*. These files might be HTML, CSS, JavaScript, JPEGs, MP3s, etc.

Second, they can send parameterized data through an *API layer*. For instance, a web page for an online shoe store might request data from an *endpoint* (that is, a specific URL) that provides a list of shoes that are currently on sale, and then a separate request to a different endpoint to retrieve data about the customer's shoe size and color preferences.

For both types of resources, web apps should serve as *reliable sources of truth*. Identical client applications should expect to get identical results, given identical requests to the server. Client applications should also be able to assume that data provided by a server is consistent and correct.

**Web servers and clients speak HTTP to each other**

Servers and clients communicate via the [HTTP protocol](https://www.tutorialspoint.com/http/). Although there's more to know about the HTTP protocol, here we want to describe what the request-response cycle looks like.

First, a client makes a request to an HTTP server. The client opens a connection to the server and sends a *request* message.

At a minimum, the request message contains the request method (GET, POST, PUT, PATCH, or DELETE), the HTTP protocol and version being used, as well as the the host (e.g., www.example.com). The request may also contain a body if, for instance, the user is posting a form or uploading a file.

Here's how an HTTP request for www.thinkful.com/courses would look:

GET /docs/index.html HTTP/1.1

Host: www.example.com

Accept: image/gif, image/jpeg, \**/\**

*Accept-Language: en-us*

*Accept-Encoding: gzip, deflate*

*User-Agent: Mozilla/4.0*

*(blank line)*

The server *response* to the request consists of headers and a body. The headers contain metadata about the response, for instance, the content-type of the response body, and the [status code](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes) of the response (for instance, 200 in the case of a successful GET request, or 201 in the case of a successful POST request). The response body contains the requested resource — in this case, an HTML page.

**Web servers route requests to the right request handlers**

This may be obvious, but it's important enough to be explicit about it: web servers need to have a way of ensuring that when a client requests a resource through a URL, that request gets routed to the right code that knows how to handle it.

**Server-side programming is about data persistence**

Although client applications can store data (for instance, via local storage or cookies) locally, web servers store the common set of data that all client applications can access. Concretely, this means that web servers house the database layer.

**Server-side programming is about business logic**

Across the layers of a server-side web application, the overriding purpose is to implement one or more pieces of business logic. Business logic should drive decisions about URL naming schemas, database queries, etc.

**Server-side programming is about security**

Many if not most of the web servers you'll work on will support user accounts. As soon as you're storing user email addresses (let alone passwords and personal data), security becomes an important concern, especially *access control*, which is how a system ensures that its resources are accessible only to the *right* users.

**Server-side programming is about software testing**

Even small server-side programming projects become complex relatively quickly, as the raw amount of code, and interdependencies between different parts of code, grow.

Alongside making effective architectural decisions, automated *software testing* is perhaps the most powerful tool we have for managing this complexity. A software test is a piece of code that checks whether or not another piece of code behaves as it should. As code is written and refactored (that is, revised), suites of software tests can be run to let us know if our changes have broken any of our existing functionality. If tests are failing, we know that the changes we've made have broken some part of our web app, and we can fix these problems before releasing into production.

**Server-side programming is about DevOps**

To achieve even basic proficiency at server-side programming requires basic knowledge of *devops* ("software **DEV**elopment" + "information technology **OP**eration**S**" [source](https://en.wikipedia.org/wiki/DevOps)). This means that as server-side programmer, you'll need to know how to do things like provision new Heroku instances, set up continuous integration, set up and maintain databases, and use version control.