Handout

Your First Meteor 1.2 Application
Lesson 5 - Step 2
David Turnbull

sitepoint premium

Introduction

Before we continue, I want to demonstrate something. You don't have to fully understand what we are about to talk about, but do follow along by writing out all of the code yourself.

Inside the JavaScript file type

```
console.log('Hello world');
```

What I didn't mention last time though, is that as a result of this statement, something else is also happening. Because if we switch away from the browser, we can see that the hello world text is also appearing inside the command line. This is significant because we've written one line of code that's being executed in two places. The code is running on the client from inside the user's web browser and on the server, where the application is hosted.

There's a few reasons why this matters, but here's one example. Ever since we created the `PlayersList` collection, the statement we see has been running within the web browser and on the server. But the code doesn't do the same thing in both places. When the code runs on the server, a collection is created inside the Mongo database. This is where our project's data is stored. But when the code runs inside the user's web browser on the client's side, a local copy of the collection is created on the user's computer. Because of this, when the user interacts with the database, they're actually interacting with a local copy. This is partly why Meteor applications are real time by default. Users can interact with data on their local machine, which can happen instantaneously, and that the changes to that data are invisibly synced in the background with the database on the server.

Running Code in Two Places at Once

At the same time though, we don't always want our code to be running in two places at once. If, for instance, we were to write code that only affects the interface on an application, it wouldn't make sense for that code to run on the server. To account for this, we can use conditionals to control where the code is executed. You'll have a much better idea of when to use these conditionals as we progress to this course, but again, for the time being, just follow along by writing out all of the code. Create a conditional that tests whether or not `Meteor.isClient` returns `true`:

```
if (Meteor.isClient) { console.log('client'); }
```

This conditional allows us to specifically execute code on the client from inside the user's web browser. Switch to the browser and notice that the message appears in the console but does not appear inside the command line. This is because the code is not being executed on the server.

We can achieve the opposite effect with a `Meteor.isServer` conditional:

```
if (Meteor.isServer) { console.log('server'); }
```

If you're having a difficult time grasping any of these details though, don't worry. You just need to remember three things:

- A single line of code can run on both the client and the server.
- This code can behave differently depending on the environment.
- Sometimes we don't want our code to run in both places. The precise moments where we need to consider these details will become obvious and make a lot more sense in the coming videos.

For now, delete the `console.log` statements from the JavaScript file, but leave the conditionals where they are.