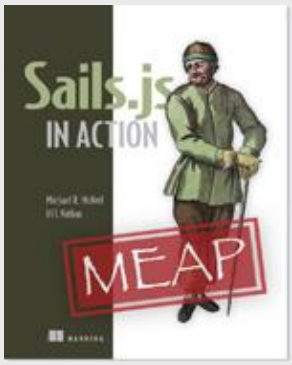

What is Sails?

By Michael R. McNeil

Sails makes it easy to create the back-end server that your front end needs to achieve an application's purpose. In this article, I talk about how Sails does this.

Sails makes it easy to create the back-end server that your front end needs to achieve an application's purpose. Those services can include everything from delivering the front-end assets to communicating with a database to interacting with 3rd party services. More importantly, Sails automates repetitive tasks found in every application and combines them with a powerful set of tools that makes executing your app faster, repeatable, and more reliable.

But what is the back end of a web application and how do you distinguish it from the front end? Let's start with a portion of a hypothetical, web application – the login screen.

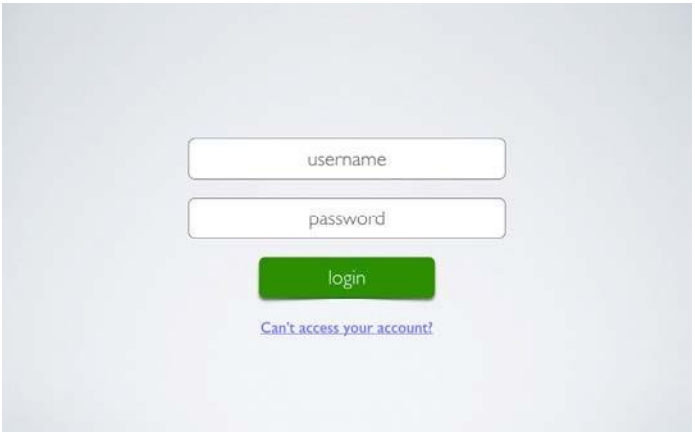

Figure 1 A hypothetical login screen.

The purpose of our login screen is to allow users to prove that they are who they say they are by providing two pieces of information to the back end to perform authentication. The front end is responsible for recognizing whether the login button has been clicked or touched

and sending the username and password to the back end. Once clicked or touched, the front end will make some type of HTTP or Socket **request** and wait for a response from the back end server telling it whether the information was legit.

As illustrated in Figure 2, a web application simply consists of a **front end** that makes **requests** of a **back-end server**. In turn, the back-end server responds to those front-end requests with whatever was requested.

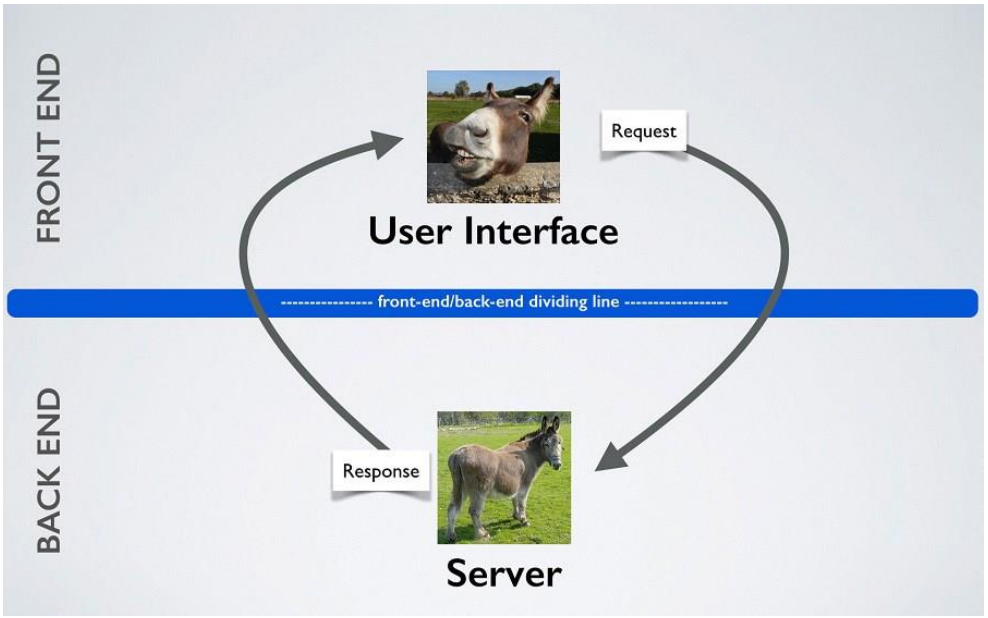

Figure 2 A web application using a simple request and response

As you can see from Figure 3, a bunch of things are going on when the front end makes its request and the back end responds accordingly.

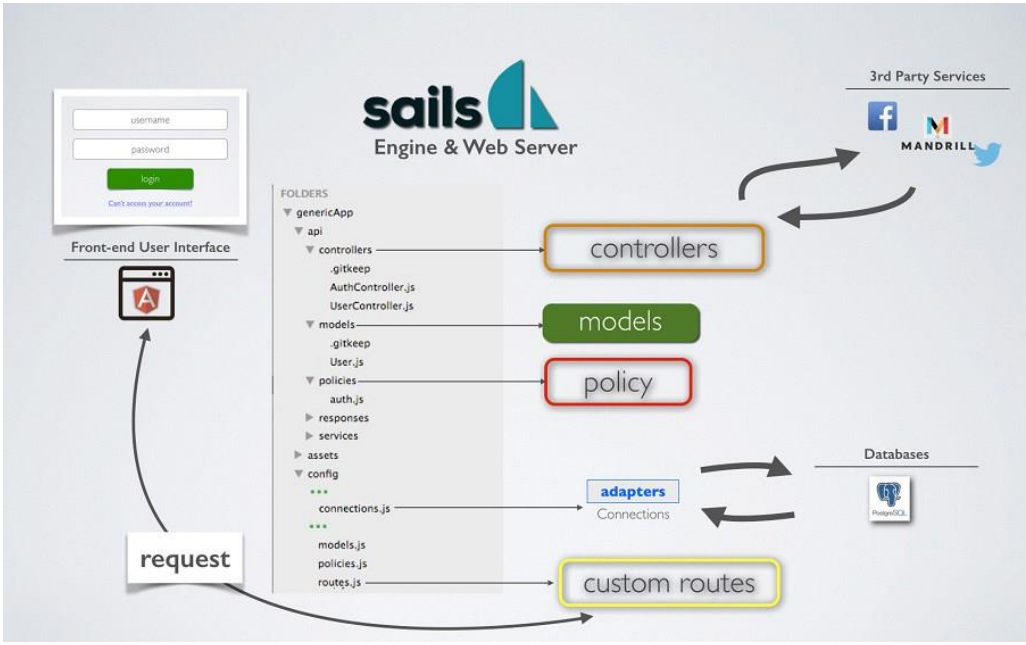

Figure 3 The login screen back end flow.

The request triggers things like routing, custom code in controllers and actions, policies that restrict access to only those you want to have access, database queries, and so much more. Sails makes creating all of that stuff a lot easier. (My book, Sails.js in Action, tells you everything you need to know to do it.)

Now, we'll do a quick tour of the major components of Sails and where they fit into web application creation as we explore this login screen. So the first question or determination we need to make is whether the back-end server will be responsible for delivering the front-end user interface? We'll address this question in the next section.

Devices dictate how a front end will be delivered

Depending upon the front-end device, Sails may be responsible for delivering the assets of that front end. Figure 4 illustrates the broad categories of *front-end user* interfaces commonly employed in web applications. Let's take a look at each front end from the perspective of how it's delivered to the device.

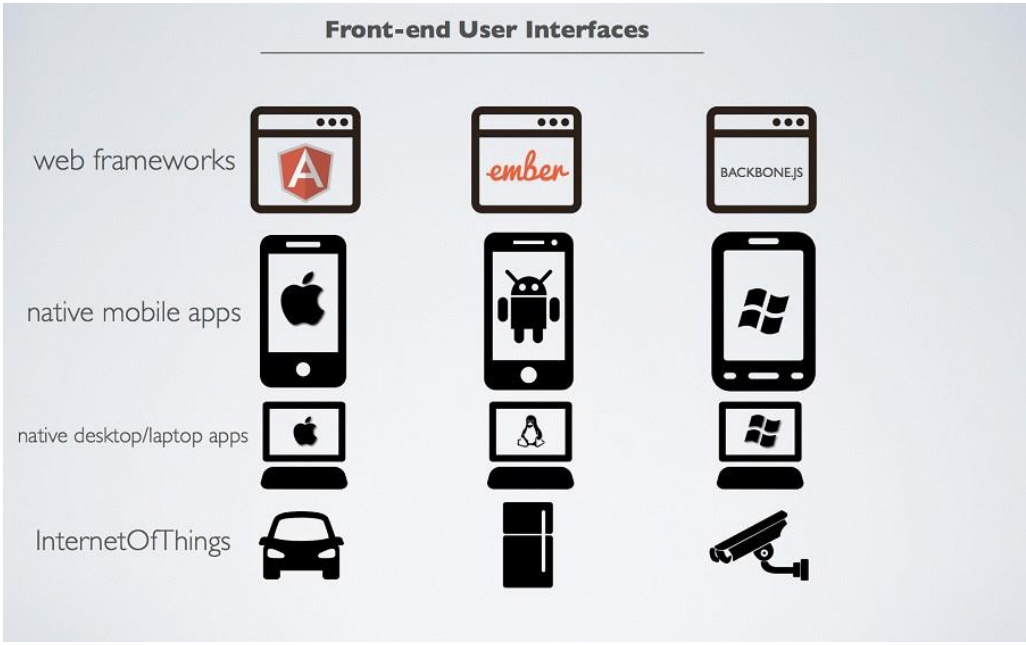

Figure 4 Front-end user interface possibilities.

Web front-end frameworks

Web front-end frameworks are typically delivered to a browser that renders them. Thus, the web front-end framework's assets (e.g. HTML, CSS, and JavaScript) rely upon a back-end server to deliver them to the browser. Sails provides a **static web server** to deliver those assets unchanged to the browser.

> You're probably already very familiar with static web servers, whether or not the term sounds familiar. You may have used tools like Apache or nginx for this purpose in the past--or if you're using a mac or PC, you may have downloaded a tool called MAMP or WAMP which does all that for you in development. In the rails world, the static web server is called Rack; in the Python world it's WSGI, in Java, it's Tomcat.

If Sails doesn't deliver the assets, what will? In production, an option is to use a content delivery network (CDN), which offers the advantage of deploying your front-end assets closer to the devices that access them. The important takeaway here is the separation between delivering the front end as one necessity of a back end from the eventual requests that are made by that front end to a potentially different back-end application. So a *back-end* framework like Sails is completely separate and distinct from *front-end* frameworks like Angular, Ember, and Backbone. In the end, the good news is that they're both designed to play very nicely together.

Mobile application operating systems as front ends

The second row of Figure 4 shows some of the most popular types of mobile app operating systems. Unlike front-end web frameworks, most mobile apps do not use a browser to render their buttons, navbars, and other user interface components. Therefore, the mobile app does not rely on the back end to deliver itself to the user, and instead, users might install the assets from the App Store or Google Play. Just like front-end web frameworks, mobile apps make requests to the back end application you'll create with Sails, which processes those requests and responds accordingly.

NOTE: Hybrid native applications, using tools like Phonegap, are rendered in an embedded browser (or "webview"), but also have access to native-level functionality, like the camera.

Native desktop/laptop applications and the internet of things (IOT) as front ends

The third row of Figure 4 represents some, but not all types of native desktop/laptop apps for operating systems like OSX, Windows, and Linux. Similar to mobile apps, desktop/laptop apps bundle their assets. That means the front-end code used to render buttons, tooltips, and other user-interface components is already installed on a given device. So these types of front-end apps don't use a browser and therefore don't rely on the *back* end to deliver their assets either.

The fourth row of Figure 4 represents smart devices that fall under the category of the Internet of Things (IOT) like smart cars, appliances, and devices (i.e., video cameras). This category of front end can use a combination of front-end web frameworks, native mobile apps, and/or desktop/laptop native apps to deliver *the front-end user interface*. Again, what is common to all of these front ends is the need for a back-end application to which they can make requests.