Naming Conventions Up and Down the Stack

Posted At : March 12, 2019 10:00 AM | Posted By : Jeffry Houser
Related Categories: CodingConventions, Professional

I recently had the opportunity to work with a brand new team building a product from scratch. I never realized how unusual this situation is until I was doing it. As a consultant, I am used to dealing with an existing code base, and do not partake in the arguments around coding conventions that occur before we start writing code. I usually step into existing project, learn the conventions quick, and start contributing. This time, I was center to the debate.

This post will focus on naming conventions, covering every piece of the stack, starting with the database, to the services layer, to REST URLs, to front end code conventions.

As the team created our conventions, we kept passing around links to random blog articles from unknown people about the way things should be. While these articles were often interesting to digest, they were always presented as a canon solution, when in fact they were just the opinion of some random guy on the Internet.

I'm writing this so that next time I can be that one random guy on the Internet whose opinion matters more than yours.

The Database

Let's start with the database. A database is the data store behind your application, and the bulk of projects I work on still use Relational databases, so that is my context. I don't have an opinion on NoSQL databases yet.

The Way it Should Be

Table names should be in ProperCase, sometimes called PascalCase. In ProperCase the first letter of a word is in uppercase, while each consecutive letter is in lowercase. For compound word, the first letter of each word should be in caps.

Here is a simple database structure:

Notice that the table names are plural. This is because every table will contain many items. For example, the Users table has many users. Each row represents a single user, but the table, as a whole, represents all users. You can see ProperCase in action in the linking table, UserRoles. The 'U' from User is in uppercase, and the 'R' from Roles is in uppercase, but all other letters are in lowercase.

Column names should be in camelCase. The camelCase syntax is just like ProperCase, except that the first letter is lowercase instead of uppercase. You see this in columns such as firstName and lastName on the Users table.

The columns should be named singularly. We might state this as an entities table contains many

entities, and each entity row has a single column. Populate this sentence with real tables from the diagram: A Users table contains many users, and each user row has a single firstName.

Stored Procedures and Views should use ProperCase in their name, because these are top level elements, just like Tables. Triggers or indexes should use camelCase because these are secondary level elements just like column names.

Why Use Something Different?

Overall, the database naming techniques I love are universal in most projects I've worked on. I did run into one situation where we decide to make the table names singular instead of plural. The reason for this is that the ORM we were using on our services layer generated class names based on the database tables, and we wanted our class names singular instead of plural. Some ORMs, such as hibernate allow us to create a mapping between database table and Java class, but others such as JOOQ do not.

The Backend Services

A service layer is built primarily to communicate with the database, but sometimes it can communicate with other services, such as a Facebook or Twitter API for example. How do we apply naming conventions to this server-side code?

The Way it Should Be

Code is put inside files, and for file names I like to use ProperCase, just as I did for table names in the previous section. Class names mirror the file names, but of course do not have an extension. So, a User.java file would contain a User class.

If I were to model the database structure from above as classes, it would look like this:

The Users table parallels a User class. The Roles table parallels a Role class. Each class instance represents a single record from the database table. The intersection table of UserRoles turned into an array of Role objects in the User class, named roles.

The class names are singular. Every class instance represents a single user, or a single role. A User has a firstName, not Users have a firstName. Property names are in camelCase, which again is just like ProperCase but with the first letter in lowercase instead of uppercase. You see the compound word affect with properties such as firstName and lastName, mirroring what we did at the database level.

The plurality of properties depends upon what the role represents. Since a user has many roles, the name of the roles property is plural. But the User can only have one firstName or lastName, so those properties are singular.

I use similar conventions for files that do not mirror database objects. File names and class names are put in ProperCase. Variables and methods on those classes would be in camelCase. You may have a UserUtils class, for example, that contains a special function to check a User's credentials. This method would be in camelCase structure, one example may be isUserInRole().

Why Use Something Different?

This approach has been consistent with most server-side technology I've had to deal with, from Java to PHP to ColdFusion and even back in the Lotus Notes and iCat days (<-- they were real technologies I swear). I have used this same approach with some NodeJS server-side projects. I have seen an alternate naming approach used on some server side NodeJS projects, but I'll discuss that when I talk about UI code and Angular in a bit.

REST Services

A REST service is a way to load data over a URL. The first time I used anything close to a REST service in a browser application, I loaded data in a hidden iFrame and wrote my own JavaScript to process it and redraw it on the screen. Things are different today.

I have seen a lot of morphing of REST services over the past ten years and the best approach seems like a moving target. I do see some convergence in ideas, and some conventions are growing on me as standards. One example is the use of HTTP Verbs to represent the action being performed. For example, GET requests are used for retrieval, POST requests are used for creation, and PUT Requests perform updates. I'm digressing. The use of these transport mechanisms are unrelated to naming conventions of the REST URLs.

The Way it Should Be

The server name, or domain name, of a URL is never case sensitive in a URL, but the rest of the URL might be, depending on the server implementation. The URL should contain the element being interacted with. Let's look at a sample URL to load all users:

```
myserver.com/Users
```

After the domain name, we have the element which is being acted on, in this case Users. A request here would process all users, most likely a GET request would return users. I specifically used ProperCase in the URL to represent Users, and this mirrors how we named the data in the services layer and the data layer. Notice I made users plural, because this request processes all Users.

To update a single user, I recommend creating a singular endpoint:

```
myserver.com/User
```

This processes a single User. A POST request would create a new User, and I would expect the body of the request to include a JSON payload representing the new user's data.

A GET request would load a single user, most likely with a URL variable, to specify which user to load, probably like this:

```
myserver.com/User/1
```

Would load the User with a userID of 1. A PUT request to the same URL endpoint would update the user with a userID of 1, based on the JSON payload represented in the body.

Why Use Something Different?

I like to make the element in the URL singular if it is operating on a single element, or plural if it is operating on multiple elements. This makes logical sense to me. However, I've seen the argument that the URL should always be plural. A URL like this:

```
myserver.com/Users/1
```

Means you are operating on the Users collection where the userID is 1. A PUT would update that User where userID is 1, and a GET would retrieve that User where the userID is 1. I personally feel this approach is less self-explanatory when reading a URL. It makes me think too much.

However, one benefit to this approach is that you'll implement all the HTTP verbs for the same endpoint. When separating out operations on a lot of Users and a single User, it is unlikely I'd implement a PUT or POST request at the Users endpoint, because we will not be creating or updating multiple users in one call. Not having two similar endpoints, with different plurals, makes the programmer need to think less about which end point they need to access and that is a positive good.

Either approach is fine as long as the documentation is there to support it and guide your REST consumers.

User Interfaces

For most of what I do today, the User Interface code includes HTML Templates, JavaScript or TypeScript files, and CSS Files. I haven't done enough mobile development, or desktop application development to have developed a strong pattern about naming conventions in those areas.

The Way it Should Be

When writing UI code, such as JavaScript my first impulse is to model the same structure that I use in the service side. File names and class names follow the ProperCase convention. Variables and methods inside a class use camelCase. I like my UI code side conventions to mirror the server-side conventions as much as possible.

Why Use Something Different?

I have been doing a lot of Angular development as of late. When doing Angular it makes a lot of

sense to use the Angular CLI, and the Angular CLI uses a file naming convention called spinal-case. Everything is in lowercase, and compound words are separated by dashes. A class might be something like this:

```
user-role.ts
```

The class inside that file would still be named UserRole, so class names are left untouched. Angular components are split up into multiple files: a CSS File, a Test File, a HTML Template file, and a spec file. The Angular CLI also specifies the type of file as part of the file name, .components for components. Creating an Angular component will give you this:

- user.component.css
- user.component.html
- user.component.spec.ts
- user.component.ts

The class inside the user.component.ts will be UserComponent. Angular uses a similar approach when creating a service, replacing the .component with .service in the file name. Services also do not have CSS or HTML files associated with them.

When building Angular applications, I recommend you use the conventions laid out by the Angular CLI, even if they may be different than guidelines on other aspects of your stack. File names aside, I still use camelCase for functions and variables that are created as part of a class.

Miscellaneous

I wanted to talk about Acronyms, ID fields, and constants a bit, because they are areas where things can digress.

Acronyms

There are two common approaches I've seen to using Acronyms within a database table, class name, variable or method. The first is to put it in all caps, this is my preferred approach. In real world text, acronyms are written in all caps, and our code should mirror that approach. Let's use the acronym TLA--Three Letter Acronym--as a sample.

You might name a class like UserTLA, a variable like this userTLA, and a file name like this UserTLA.extension. If the acronym comes at the beginning instead of the end, I'd name a class like TLAUser, a variable like TLAUser, and a file like TLAUser.extension. This breaks other conventions for the variable name, so you might call it tlaUser, but that is never my preference.

The second approach is to cap the first letter and lowercase the rest. You might name a class like UserTla, a variable like this userTla, and a file name like this UserTla.extension. If the acronym comes at the beginning instead of the end, I'd name a class like TlaUser, a variable like TlaUser, and a file like TlaUser.extension. For variable names, this also breaks the convention of lower casing the first letter of the variable, so sometimes I'll see something like tlaUser. I'm not a fan of using any

lowercase in acronyms. Although it may offer more consistency across the code base, than putting the acronym in all caps, it still forces my brain to think harder when reviewing code.

ID fields

An ID field is a special case. ID is not necessarily an acronym, it is more of an abbreviation for identifier. It is common in the database to add the ID postfix to the column that uniquely represents a record--we call this the primary key. I prefer to put ID in all caps, such as userID. Others prefer to capitalize the I and lowercase the D, such as userId. A recent project is using the second convention, lower casing the `d`. It has taken me 6 months, but I've started to reverse the muscle memory I've built up over the years.

Constants

Constants are just like variables in code, but the intent is that they never change. To distinguish constants from regular variables, I often see them created with all caps. For compound words, I separate them with an underscore. A constant might be created like this:

```
NEW_USER_ID = 0;
```

An underscore separates multiple words, and this process can be called Snake_Case. This approach seems pretty universal in all technology I have worked with, but I've never seen it required.

Final Thoughts

I want to leave off with this quote from the Google Style Guide:

> "Every major open-source project has its own style guide: a set of conventions (**sometimes arbitrary**) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style."

The emphasis is mine. Conventions are arbitrary. At the end of the day, the naming conventions you use on your project do not matter. I like to encourage consistency because that it makes it easier to maintain the code base, for yourself, your future self, and your replacement. But the actual choices you make are more a matter of personal preference than anything else.

More Reading

- Microsoft Design Guidelines
- Google Style Guide
- Apple Swift Naming Conventions
- Angular Style Guide
- CSS Naming Conventions
- Swift Style Guide

- JavaScript Conventions