The MVC Design Pattern in Vanilla JavaScript

Help us help you! You'll get a ...
FREE 6-Month Subscription to SitePoint Premium
Plus you'll get the chance to
WIN a new Macbook
SitePoint 2017 Survey
Yes, take the survey
No, thanks

Design patterns often get incorporated into popular frameworks. The Model-View-Controller (MVC) pattern is one such pattern. Many JavaScript frameworks implement MVC in some form or another. Frameworks borrow from the design pattern. Oftentimes, a particular framework will come with its own interpretation of this design pattern. Frameworks come with opinions and each one forces you to think in a certain way.

Modern frameworks discuss what the concrete implementation of the MVC pattern looks like. This could be a data binding system, a template system, or even which side code and views. When we only look ways those from one framework, we don't see the pattern itself. We don't see what it looks like in a single, isolated context.

More from this author
A Guide to Proper Error Handling in JavaScript
Common Pitfalls Implementing ASYNC Ajax Apps

In this article, we'll look at how to implement the MVC design pattern in a decoupled approach in JavaScript. We'll see how to implement a simple, isolated, and decoupled MVC. We'll see how to implement a design pattern separate concerns and promote clean code.

A framework-agnostic approach to MVC is adhere to a design pattern, which is not a framework from a pattern is to look for the hollywood principle. The hollywood principle is "don't call us, we'll call you". Anyone there is a dependency that calls the framework. Instead, the framework calls the code. The framework is in the driver's seat. It calls the code. It is in the driver's seat. Instead, we want the code to be in the driver's seat.

There are good reasons to avoid client-side frameworks:

- Frameworks add complexity and bloat to a project
- Frameworks can be overkill for simple, non-interactive/complex code
- As new features are added, it is difficult to remain existing legacy code

The MVC Pattern

The MVC design pattern emerged from the Xerox PARC research project in the 1970s and into the 80s. It is a pattern that is used in many programming languages. Many JavaScript frameworks implement MVC. Many JavaScript libraries applications for has proven to be effective for web apps for.

At its core, the MVC design pattern is about clear separation of concerns. The idea is to make the solution intelligible and mining, like Nokia programmer looking to make specific changes can find the right spot well.

Read Modern JavaScript
Keep up-to-date with the evolving world of JavaScript
A MODERN JAVASCRIPT
Start Reading Now

A Demo of Penguins

Let's consider a simple application that displays information about penguins. The demo, for, then, an IT Infinite Bank of penguins which start at an X-branch connection.

We'll be a demo of penguins for a web showing several species on a single page. For this, I'd like to use the MVC design pattern. The MVC pattern is good for clean code and for separation of concerns. It promotes code that is modular and easy to maintain. It promotes. At the end, you should be able to go through a few penguins each with its own data and profile section.

By the end of this article, you should be able to implement a simple MVC design pattern in plain JavaScript. The pattern could be super lockable or expect good and robust.

I'll show a DOM-like API for the browser's document.querySelector. It makes sense to use present-language features with the presence design pattern.

Are you ready? Let's find out.

The skeleton

The skeleton consists of three main parts: controller, view, and model. Each with its own concern and problem that it needs to solve.

A visual of what this looks like is below:

The PenguinController handles events and is the interface between the view and model. It reacts to user actions and requests when the user clicks on a penguin. It then requests the data from the model and passes it to the view. The controller knows when to ask for data and when to ask the view to update. The controller is the only piece that knows about the rest of the system. It is the only piece that knows about the rest of the system. It knows the rest of the system. It knows the rest of the system. It knows