Building Front-ends

Thierry Sans

Recipes to become a good front-end developer

Good practices with Javascript

- Write good Javascript code
- Encapsulate Javascript in closures

MVC design pattern (Model-View-Controller)

- The advantages of MVC
- Homemade MVC framework

Good Practices with JavaScript

The problem with Javascript interpreters

- ✓ **Good Javascript** is interpreted by browsers in a consistent way
- ◉ **Bad javascript** code is loosely interpreted by browsers in an inconsistent way

Solution 1: using **`strict mode`**

➡ Force the browser to validate Javascript against the standard

✓ Dynamically raises errors (or warnings) in the console when the code is not compliant with the standard

```
"use strict";
var doSomething = function() {
    // this runs in strict mode
}
```

Solution 2 : using `JSHint`

- ➡ Analyze Javascript source code with JSHint
- ✓ Statically finds bugs and reports them in the terminal

```
$ npm install -g jshint
$ jshint js
```

Problem with scoping

- In the browser, all Javascript files share the same execution environment i.e they share the same scope
  - variable (and function) naming conflicts
  - strict mode applied to all

Scoping problem with variable names

```
----------------------------------------
                                file1.js
var doSomething = function() {
    // first declaration of doSomething
}

----------------------------------------
                                file2.js
var doSomething = function() {
    // shadowing doSomething from file 1
}
```

Scoping problem with strict mode

```
-------------------------------------------------
                                          file1.js
"use strict";
var doSomething = function() {
    // strict mode applies
}

-------------------------------------------------
var doSomethingElse = function() {        file2.js
    // strict mode applies too
}
```

Solution : encapsulate Javascript in **a closure**

```
(function() {
    "use strict";

    var private = function() {
        // private is not available from outside
    }
}());
```

Solution : encapsulate and export the **namespace**

```
var $ = (function() {
    "use strict";

    var export = {};

    var private = function(){
         // private is not available from outside
    }

    export.public = function(){
         // public is available from outside
    }

    return export;
}());
```

Model View Controller

Model - View - Controller (MVC)

Model View Controller in Software Engineering

➡ Software architecture based on **design patterns**

Model	**Data**
View	**Presentation**
Controller	**Business Logic**

MVC - a popular pattern for web development

Most web frameworks (front-end and backend) rely on the MVC design pattern

BACKBONE.JS

METEOR

ANGULARJS by Google

express

React

ember

Advantage of MVC in web development

Separation of duties between different experts

Model	**Data**	The database programmer
View	**Presentation**	The web designer
Controller	**Business Logic**	The web programmer

Other advantages of MVC

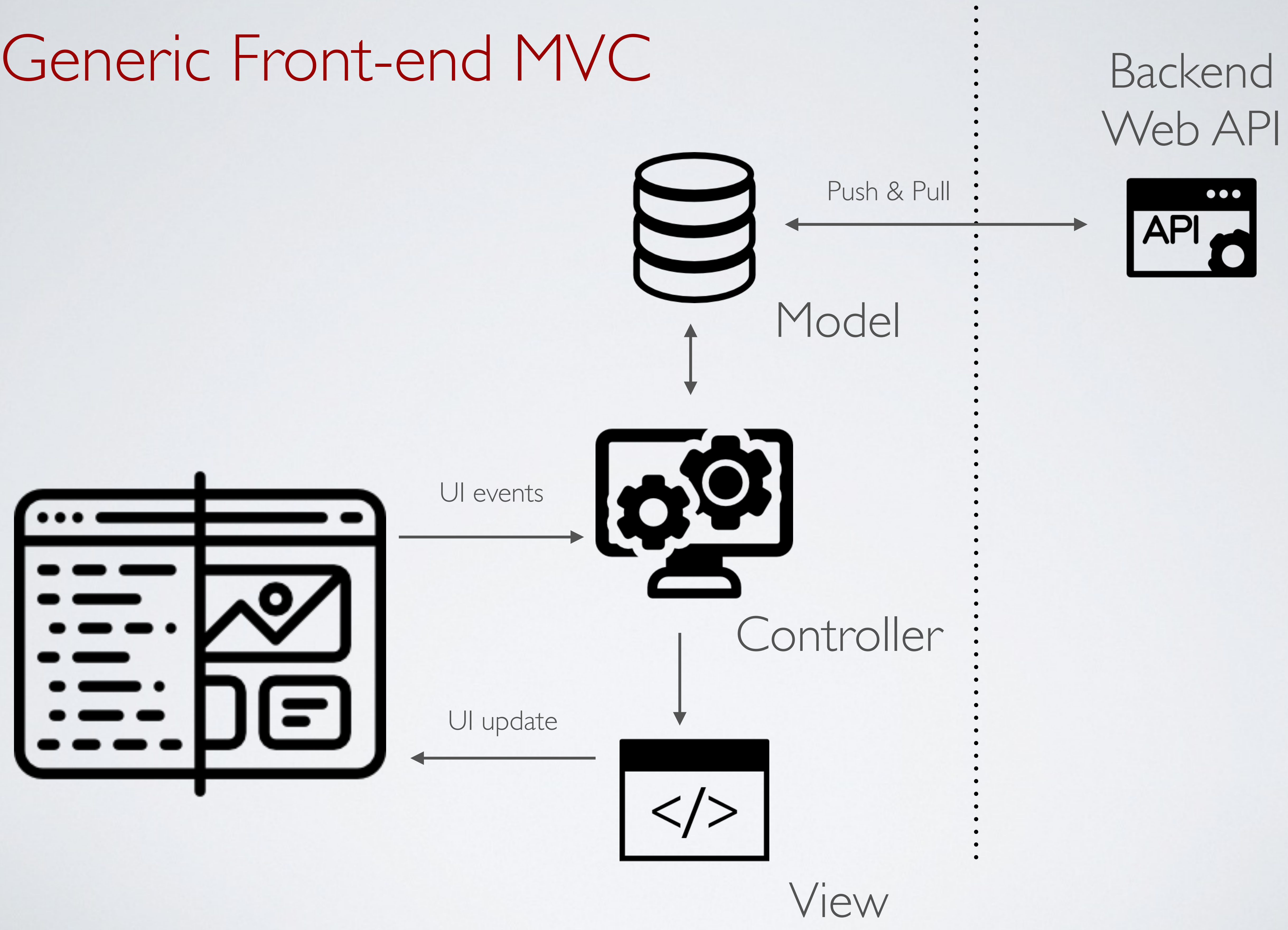
Easier to **design**

Easier to **maintain**

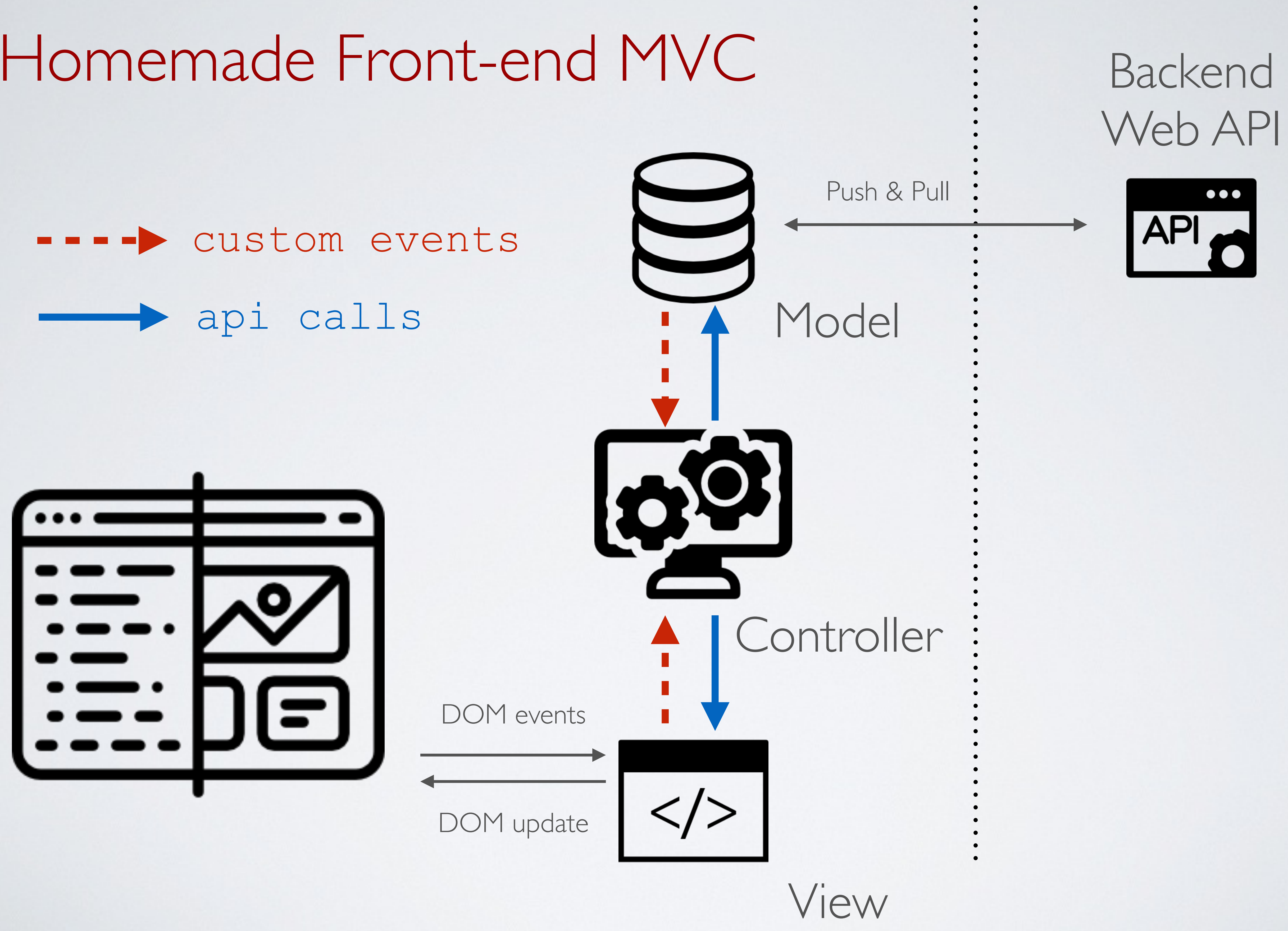
Easier to **test**

Easier to **scale**

Generic Front-end MVC



Homemade Front-end MVC



Rules for our homemade MVC

Model - deals with the data

✓ Provides an API to the controller to push/pull data

✓ Raises events when data are modified

View - deals with the UI (DOM)

✓ Provides an API to the controller to modify the document

✓ Raises events when users interacts with the document

Controller - deals with the model and the view

✓ Reacts on events coming from the model and the view

◉ Does <u>not</u> deal with data nor UI

Model

```
var model = (function(){
    "use strict";

    var local_datastore = ...

    var model = {};

    // Create
    model.createData = function(data){
        // add data to the local_datastore
        // dispatch event
       document.dispatchEvent(new CustomEvent('onNewData',{detail: data}));
    };

    // Read
    model.getAllData = function(){
        return local_datastore;
    };

    // Update

    // Delete

    return model;

}());
```

View

```
var view = (function(){
 "use strict";

 // UI events
 document.getElementById(some_form).onsubmit = function(e){
    // get the elements from the form
   // dispatch event
    document.dispatchEvent(new CustomEvent('onFormSubmit',{detail: data}));
 };

 // UI API
 var view = {};

 view.insertData = function(data){
    // update the DOM with the new data
 }

 return view;

}());
```

Controller

```
(function(model,view){
    "use strict";

    document.addEventListener('onFormSubmit', function(e){
        // get data from the view
        var data = e.detail;
        // forwards it to the model
        model.createData(data);
    });

    document.addEventListener('onNewData', function(e){
        // get data from the model
        var data = e.detail;
        // forwards it to the view
        view.insertMessage(message);
    });


}(model,view));
```