|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385330) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385330) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385330) |
|  | |
| 19. The Model-View-Controller Design Pattern  19.1. Overview  *Copyright (c) 2014, Rula Khayrallah*  The model-view-controller (MVC) pattern is a software design pattern that is widely used in web application development.  The model view-controller paradigm organizes an application into three components.  The **model** manages the **data** manipulated by the application.  The **view**deals with the specific **presentation**of the data to the user.  The **controller** contains the **logic** of the application.  It mediates the interaction with the model and the view.  **Each component has a well defined interface and interacts with the other components through that interface.**  The interaction between the components and the user can be illustrated as follows:    The user 'sees' the application through the view component.  However the user's actions are handled by the controller component.  The model component is accessible only through the controller.   The controller can send queries as well as updates to the model.  The controller does not present data to the user directly.  It invokes the view component to do so.  Consider a simple business review web application such as the one we  encountered in a previous module.  The model deals with the data associated with the various businesses.  That data includes the names, descriptions, ratings, prices, addresses and so on and is probably stored in some database on the server.  The application may include several views:  one that presents the user with the data specific to one business and another that includes some summary data about several businesses at once (a view with all businesses sorted by rating, for example).  Finally the controller is responsible for handling the interaction with a user. For example it may respond to a given user request by getting a subset of the data from the model to be presented with a given view.  Note that in some implementations of the MVC pattern, the model updates the view directly without going through the controller. In this course, we will follow the more rigorous implementation depicted in the diagram above.  There are several variations on the MVC pattern:  MVP (for Model View Presenter), MVVM (for Model View View Model) and more recently,  as illustrated with the AngularJS framework, MVW (for Model View Whatever).  19.2. Server Side or Client Side?  *Copyright (c) 2014, Rula Khayrallah*  Web applications used to follow a thin client approach where the model, view and controller resided entirely on the server.  Nowadays, it is more common to see MVC components on the client side.  Going back to our business review app, let’s say that the user is currently viewing a number of nearby restaurants.  What if the user wants to focus on the restaurants with the highest ratings and wants to view these restaurants sorted by (decreasing) rating?  There is obviously no need to go back to the server for this. Handling this request in the client side JavaScript 'controller' is easy and will result in a more responsive application.  Similarly we may have a **model** component on the client side to **manage the data in local storage or session storage or even to get data from the server through ajax calls.**  Finally, the**view**  component on the client side would be responsible for **updating the web page through DOM manipulation.**  **MVC Frameworks**  Several frameworks that enforce the MVC design pattern are available on the server side as well as on the client side.  Rails and ASP.NET MVC are popular server side (non JavaScript) MVC frameworks.  On the client side, JavaScriptMVC is an open source MVC framework based on jQuery.  Backbone.js  and Angular are both popular client side MVW JavaScript frameworks.  19.3. Advantages  *Copyright (c) 2014, Rula Khayrallah*  Even though applying the model-view-controller design pattern to a web application may introduce some overhead work and additional code, the benefits of doing so become clear in the long term and as the size and complexity of the application increase.  Here are some of the advantages of the model-view-controller paradigm:  1.   Complexity tamed:  instead of looking at the daunting task of implementing all aspects of the web application at once, we can focus on one component at a time.  2.   Easier maintenance:  changes made to one component will not affect the others (as long as the interface is kept intact.)  Maintenance of the web application becomes easier.  3.   Reuse: a controller written for one web application may be reusable in a completely different application with different model and views.  4.   Testing: the three components can be tested independently.  Stubs and mocks may be used to emulate the functionality of the 'other' components.  5.   Specialization: the separation of the components allows different developers to work on the different aspects of the web application.  This is useful when developers have specialized skills.  6.   Parallel Development:  the three components can be developed in parallel by different teams.  19.4. Example  *Copyright (c) 2014, Rula Khayrallah*  Let’s go back to our simple calculator example and implement it using the MVC design pattern.  To justify the addition of a model component, we'll add the requirement that the last numbers entered by the user be stored in the local storage.  We’ll modify our HTML source document to point to the new MVC script.  calculatormvc.html  <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>My Simple Calculator</title>  </head>  <body>  <h2> Let's add some numbers! </h2>  <p>Please enter two numbers:  </p>  <input id = "first" type="number">  <br>  <input id = "second" type="number">  <br>  <p> And the answer is:</p>  <p id="answer"></p>  <script defer src="../scripts/**addmvc.js**"></script>  </body>  </html>  We modify our initial JavaScript add.js code to separate the code into three distinct components:  a model, a controller and a view.  **Each of these components will be a JavaScript object and we’ll define methods on these objects to handle the tasks associated with them.**  Note that in this trivial example, the controller only sends updates to the model but in a more general case, the controller may use the model to get access to the data as well.  addmvc.js  // MVC implementation of our simple calculator  // The controller consists of one function: update.  // It is invoked when the user types anything in the input fields  'use strict';  var controller = {  **update:**function () {  **// Get the input from the view**          var input = **view.getInput()**;          // Then  compute the sum          var firstNumber = Number(input[0]) || 0;          var secondNumber = Number(input[1]) || 0;          var myAnswer = firstNumber + secondNumber;  **// update the model**  **model.save**(firstNumber, secondNumber);  **// update the view**  **view.showResult**(myAnswer);      }  };  // The model saves the data to local storage.  var model = {  **save:** function (first, second) {          localStorage.setItem('firstNumber', first);          localStorage.setItem('secondNumber', second);      }  };  // The view component has 2 functions: getInput and showResult  var view = {  **getInput:** function () {          var firstInput = document.getElementById('first').value;          var secondInput = document.getElementById('second').value;          // return an array with the 2 inputs          return ([firstInput, secondInput]);      },  **showResult:** function (result) {          document.getElementById('answer').textContent = result;      }  };  // Event listeners  // The controller handles all user input so the event listener here is a controller method  document.getElementById('first').addEventListener('input', **controller.update**, false);  document.getElementById('second').addEventListener('input', **controller.update,** false); |  |