UDC 004.4

I.V. Ponomarev

# DEVELOPMENT OF WEB-SITES WITH USE ASP.NET TECHNOLOGIES AND ANGULARJS PLATFORMS

*Annotation. There are many models and technologies for creating websites. Classic models are constantly supplemented by new developments. The choice of one or another implementation depends on many factors arising from the goals and objectives defined at the planning stage. Features of the ASP.NET WEB API 2.0 and AngularJS platforms are described.*

*Keywords: site, Web application, client-server architecture, model, controller, view, pattern, framework, technology, .NET WEB API 2, AngularJS, MVC, HTML.*

**Introduction.** The Web application consists of two parts - the client part that represents a user-friendly interface (UI, user interface), and the server part that provides access to the database through the API (application programming interface). To create Web applications on the server side, a variety of technologies and programming languages are used: ASP, ASP.NET, Java, Perl, PHP, Python, Ruby, Node.js, ASP.NET vNext, Coldfusion. On the client side, they are used to implement the GUI: HTML, XHTML, CSS, and for creating and processing queries, creating an interactive and browser-independent interface: ActiveX, Adobe Flash, Adobe Flex, Java, JavaScript, Silverlight.

**Formulation of the problem.** A Web application is created both on the server side and on the client side. It is necessary to consider the specifics of the development of sites based on ASP.NET WEB API 2.0 [1] and AngularJS [2].

**Main part.** Server architecture. The server part is the access point to the database. The ASP.NET Web API is the foundation that makes it easy to create HTTP services that are used to communicate with clients, including browsers and mobile devices. ASP.NET Web API is the ideal platform for building RESTful applications on the .NET Framework.

The method of interaction is simple - in the application there is a controller Controller. Each controller is responsible for its essence from



 ISSN 1562-9945

the database and working with it. For example, ClientController will be responsible for working with the client entity (creating a new one, saving changes to the existing one).

The model includes the logic for storing and retrieving data from the repository, but even within the model, you must maintain a certain level of separation between the data model entities and the storage and retrieval logic, which is achieved using the storage template.

The program operates with a number of entities and models, for managing which, a number of repository classes Repository are created. The repository allows you to abstract from specific connections to the data sources with which the program works, and is an intermediate link between classes directly interacting with the data and the rest of the program.

The UnitofWork pattern creates its own work area for each request, which avoids the problem of concurrent access to the same data. Thus, work with various repositories is simplified and it is guaranteed that all repositories use the same data context. The controller interacts with the database through a service that uses extensions of entities, updates them and passes them to UnitOfWork. UnitOfWork through Repository and object-oriented technology for working with data Entity-Framework Context works with the database (Fig. 1).

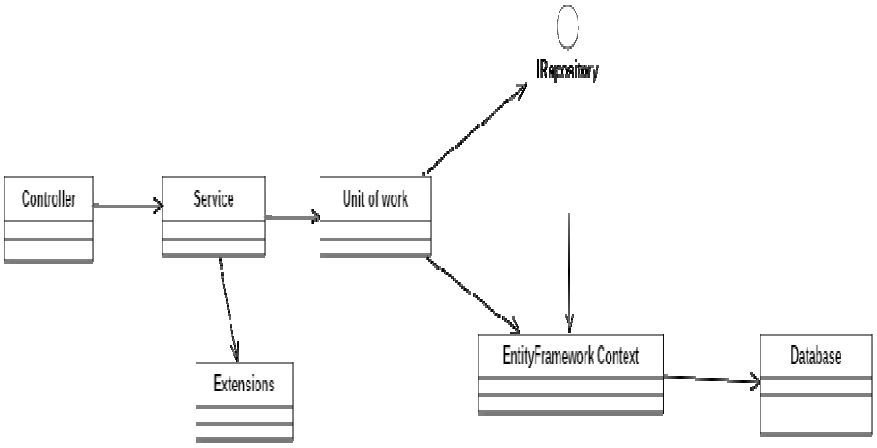


Figure 1 - Server architecture

Architecture of the client part. The client part is the layer between the user and the database. Let's consider the possibility of its construction on the AngularJS framework.

AngularJS is a popular JavaScript framework with the MVC architecture, which is widely used to create and support complex web applications.

Each page has its own controller, where minimal business logic occurs - all manipulations occur through the server, the view is an HTML page, and the model is the actual data that the user will manipulate.

Angular splits the application into blocks of several types: controllers, directives, factories, filters, services and views. Types deal with the user interface, controllers - interface logic, services communicate with the backend, directives allow you to create components and extend HTML. Blocks, in turn, are divided into modules.

To create a controller, use the controller (name, constructor),

which is defined in the Module object. The first parameter passes the name of the controller, and the second one - the function that performs the controller tuning.

To apply the controller to a specific HTML markup area, you must use the ng-controller directive. After that, this part of the HTML markup will denote the controller's view.

The $scope object serves as a link between the representation in the form of an HTML code and a controller. The $scope object acts as the program model. $scope defines any objects that you can use in the view and to which you can set data binding. Formally, $scope represents a regular javascript object.

In the controller function, the $scope service is passed as a parameter, through which the controller transfers data to the view (Fig. 2).

Each AngularJS program has one $rootScope object, which is the parent of all other $scope objects used in the controllers. And when you run the program, you create an anchor for the element that uses the ng-app directive to the $rootScope object.

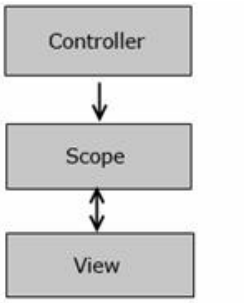


Figure 2 - Interaction of parts of AngularJS

AngularJS services represent special objects or functions that perform some common tasks for the entire application. In AngularJS, there are a number of built-in services, such as $http, $q, $timeout and

a number of others. In addition, it is possible to create your own services to perform specific tasks.

Client-server interaction is implemented using the service $http to work with http-requests:

```
$http.get('/ServerRequest/GetData', config)
.success(function(data,status,headers,config) {
        $scope.Details = data;  })
.error(function (data, status, header, config) {
        $scope.ResponseDetails = "Data: " + data +
            "<hr />status: " + status +
            "<hr />headers: " + header +
            "<hr />config: " + config;   });
```

A GET request is made to the desired address for information. The returned object is associated with two specific success and error methods. The success method is passed a function that will work as a result of successful execution of the request, and data is transferred from the server to use them. The error method will only work when the request is not executed and an error is returned. The error data will also be passed to the function.

**Conclusions.** The architecture of the server and the client part is considered, the principle of interaction between these two parts of the application is described. A web application for accounting of goods turnover, customers and orders was developed using the ASP.NET WEB API 2.0 and AngularJS platforms. To create a product page, templates of HTML-page with AngularJS directives were applied, which simplify the programming of simple interfaces. Using the AngularJS framework to develop the client part of the site allows you to create web applications of any complexity very quickly, since AngularJS contains all the necessary services for application development.

**LITERATURE**

1. Adam Freeman. Expert ASP.NET Web API 2 for MVC Developers - Apress, 2014 - 688 p.

2. Ari Lerner. ng-book - The Complete Book on AngularJS - Fullstack, 2013 - 624 p.