# How to Implement Hacker News

1. General

The goal of this document is to provide a high level design document discussing the different technologies one might use to build the website Hacker News (https://news.ycombinator.com/).

1.1. Project general description

Hacker News appears to consist of a single page application (SPA) to present text-driven information. Within the menu at the top of the page are links for the following categories: new, comments, show, ask, jobs, submit, and login. When unauthenticated, submit and login both point to a login screen that allows for authenticating and for creating a new account. After authenticating, the submit button allows for submitting content.

There are several links at the bottom of the site that point to static files, pages that are located on other sites, and e-mail addresses.

1.2. Solution general description

The solution will require a mixture of client side, server-side, and back-end technologies, not limited to the Angular framework (v.2 or higher), an ASP.NET MVC application, Web API services encapsulated within executables hosted in Azure using Service Fabric, and DocumentDB for the database to allow for easy storage, since maintaining relations is not a primary concern of the application.

1.3. Guidelines from the functional design

The responsibility of the Angular framework and ancillary JavaScript libraries will be limited to rendering data via templates identified within components and directives contained therein, client-side validation for form posting, and service calls to Web API endpoints to post and retrieve data.

The MVC application will be where the controller is implemented for rendering the SPA, as well including requisite functionality for bundling client side libraries (AngularJS and its components acquired via Node Package Manager) and style sheets using Webpack.

The different Web APIs managed using Service Fabric will be stored in their own executables, one for each page in the site, plus for authentication and authorization. E.g. the comments menu item will have a dedicated Web API with its own methods.

1.4. Development guidelines.

All classes on both the client side using TypeScript and on the server side using C# will be written in separate files for the purposes of version control.

As mentioned above, the main components are the client-side, server-side, service, and database architectures. To implement the first two, ASP.NET Core’s ASP.NET MVC SPA template will be used because of the built-in Angular, Node server, and Webpack functionality. For the Web APIs, stateless Service Fabric applications will be implemented

1.4.1. Best Practices.

For consistency’s sake, camel notation will be used in all coding. All Web APIs will implement attribute-based rather than convention-based routing. Within the individual Web APIs, an MVP pattern will be implemented so that any change of state to a particular controller may be processed within a separate class, such as for validation. Each Web API will also have its own data access layer implemented using a repository pattern for interacting with the DocumentDB database.

The models will be coded using implicit schemas within the Web APIs, not solely based on the shape of the data stored within each collection in the database. The models used within the Web APIs will not necessarily conform to the shape of any objects used within the AngularJS framework after it is serialized and transmitted to the client from any particular endpoint.

2. Architecture

The architecture will be a distributed architecture for the purposes not only of separation of concerns at the code level, but also for health monitoring and maintenance of the application, including deployments.

2.1. Logical architecture

2.2.1. Client Side

Below the app.model component class in AngularJS will be one for each of the menu items: new, comments, show, ask, jobs, submit, and login. Each one will have separate files for their respective templates, components, and service classes. There will be a directive available for each of them to be identified in the index page, which will serve as a template to format content provided by the different components.

Each service class for the different menu types will have at least three methods for calling a Web API service, one for getting a list of data, one for getting data for a specific item in a list, such as a comment from the comments page, and another for posting it.

For the methods getting data from a service, there are a few models: The class containing a list, a class conforming to a list item, and a class containing the data for what appears on-screen after clicking on the link for the list item.

The list class contains a property for an array of the second class, which contains properties for the username of a commenter, the time stamp of the posting, the id number of the parent id of the comment, and the id number of the comment to see it in its entirety.

The third class, which contains the data for the comment, includes properties for the following; an external link, points acquired by the commenter, his username and id number, and a collection for storing comments on the user’s comment. That collection will be made up of a comment model which will include the comment’s formatted text, the comment’s user, the time stamp of the comment, and a collection of subsidiary comments.

For the method posting data from the Angular service, there’s a model that will capture the user’s the text from the form the user filled out to comment. The method implements a call to a Web API service for acquiring tokens for authentication to get a token that enables the RESTful call to post the new comment.

Each of the menu components’ client side architectures conforms to this paradigm.

2.2.2. Server-Side

The server side architecture within the ASP.NET MVC application does little other than render the SPA application and store the JavaScript files that enable AngularJS to function.

2.2.3. Services

As mentioned above, there are separate Web APIs contained in their own executables that send and receive RESTful data with the web application. The services will generally have three or methods that they use to transmit data asynchronously with AJAX calls from the client in a serialized JSON format. All will use attributes to define how they are routed to by URL, HTTP verb, and signature. The latter will be mapped against models defined within the apis that will include properties such as strings for usernames, long integers for ids, and strings for comments.   
  
For posting data, OAuth will be used to authorize rights in tandem with a separate service that provides the token for the handshake. Validating, such as checking for dangerous code, happens using attributes on the action methods in the api’s controller. Other possible processes take place in the presenter class in the MVP pattern, where appropriate.

Below the presenter class is the repository pattern used to map data against the collections in the database. Here is where the models are implemented to provide implicit schemas for various CRUD operations. For each of the pages, the models tend to be fairly similar.

3. Database

As mentioned before, DocumentDB is used for the collections that store information for each page in the website. There is a collection for each page type: new, comments, show, ask, and jobs. Since the information they contain is textual and rather amorphous, the collections are ideal for this kind of storage. All comments associated with a comment will be stored as BSON within the same record.

For authentication purposes, username and passwords will be kept in a SQL Server database in order to take advantage of the simple membership tables that are readily implementable and allow for features such as storing usernames and passwords, password recovery, failures to authenticate, et cetera.