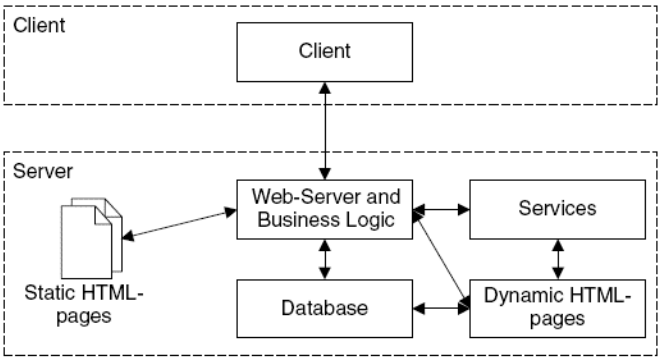
Chapter 4. Layered Architecture for Web Applications

Table of Contents

The first question that should be answered: Why the web is suitable for developing applications? It is not a difficult question, of course. In the World Wide Web Consortium (W3C) Architecture of the Web Recommendation paper various examples are given, notably one about a user who wants to see the weather in a place where she wants to travel to. The nature of the Internet - trafficking data over protocols from network to network - is a powerful resource that make communication between different places very fast and easy. This way computer programs can be made to improve the relation between systems and people, and what is seen today is that it happened.

Web applications are distributed applications, and hence are at least two-tiered. They act as a special kind of client-server applications, where a large portion of the functionality is "pushed" back to the server side despite the fact that the Web does not define what is behind the server.

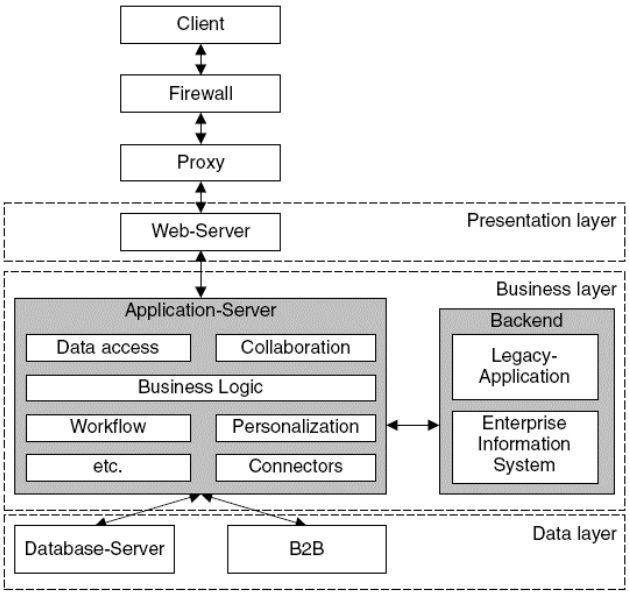


The basic two parts for Web Applications

The Web relies strongly in the client-server model, and it uses markup languages such as HTML and XML to transfer and represent data. Under it there are many programming and scripting languages that can dynamically process, modify and generate data, or give an user interface. This way, the development of Web applications can be put under the cover of software engineering but need to be extended. Web applications are multidisciplinary (software engineering, database modeling techniques, network computing, and effective interface design). They are built in a continuously changing environment where requirements are unstable and the user community is wider than before. Web applications handle information from various sources (text, graphics, video, audio) dealing with structuring, processing, storing and presenting this information.

The Three Layers Model

The nature of the Web is layered: it has formats over protocols and uses a client-server model. Therefore, it is natural that a layered architecture would be suitable for developing to the Web. We learnt that this model overcame the two layered client-server because of its scalability. Many different approaches to the aim of developing applications with different layers had been used along the years, but a clear pattern seems to appear frequently in various of them: the Three Layers Model (according to Kappel et al. 2006).



The standard three layered architecture for Web Applications

This model of web application development is very similar to the Service Layer/Domain Model/Data Source Layer set of design patterns from Martin Fowler's collection, but receiving different names . In fact, the idea (usually named 3-tier architecture, or expanded into n-tier architecture) is very general and widespread, so in this paper only the most common assumptions and uses are examined.[1]

Its conception is three layers, one over the other, being the application the set of them working together. The most external of them is the View Layer, that is the visible part of the application, the one that interacts with the user. The layer in the middle is the Business Logic Layer, which serves as an intermediate between the View (or presentation) and the innermost layer, that is the Data Layer. This last one is where all the data used by the application is stored.

The benefits of using layered models in software development are in the way that it is easier to know exactly what each part of the application does. It makes easier to construct the application, to debug it, and to maintain and reuse the code. For example, if it is needed to exchange some details in the presentation of the content but the business rules and data models do not change, only one layer is affected. Even for more complicated changes involving all of the application architecture there are benefices, so a plan can be created in the overall but specifying exactly where the changes need to be done.

The View Layer

The outermost layer in this kind of model deals with the presentation of the content and interaction with the user. It can be called view, presentation, UI. In this layer the application shows to the user what is needed to be seen and gives the tools for interaction. The exact kind of interaction depends on the application; one can create a web app that only shows information to the user without any kind of interaction, not even hyperlinks to be clicked, but such a case does not need an advanced architecture. In most cases the user will generate some input, send it for processing and then receive a feedback, that can be the final result or a step for further operations.

Following the example by W3C of the user that wants to see the weather in her trip destination, the presentation is where she sees the actual content. The content display is shown, and the user can interact with the provided controls to, for example, see weather in different periods of time or another places, and see pictures of it.

The technologies usually involved in this layer on the web development context are mainly the markup that is processed by the browser (HTML/ XHTML/ ...), the style of the page (CSS) and client-side scripts (Javascript/ Flash/ ...). All of these tools together can produce a rich environment for user interaction and content display. Of course it can be said that server-side scripts can be used to generate content, but at the final level these scripts produce the HTML that will be shown be the browser, so this role of the development can be subdivided: the content generation is created by the business logic layer (the next topic to be discussed) and then it is passed to the view layer, maintaining the logical division of the application. The browser shows the content initially written or produced by the server-side scripts, and the client-side scripts are able to modify that content. A Javascript code, for example, can be used to validate form data or even to create drag and drop interfaces. It interacts with HTML through a DOM tree, that is a representation of the document in memory. HTML5, the present (2014) trend for web development is praised for its flexibility, specially where it touches the concept of responsiveness, that is the ability to change the content disposition according to the screen size. This matters because, in current days, the availability of a page in different screen sizes and devices is extremely important. Having many possibilities like desktops, tablets, smartphones, wearable devices and even augmented reality or voice user interface, the range of technologies and targets for the view layer is very wide, and it shows both the importance of it to the user and reinforce the need of a logical division of the application for supporting such variety.

This layer communicates with the business logic layer under it, passing the information from the user and controlling it, then giving back any response it produces, not leaving any decisions of the application's logic to be resolved by the UI. This passing of information is usually done through forms, like a user log-in in a system by giving username and password, but there are other ways. AJAX is an asynchronous way to pass information to the server and get responses. The cited a-synchronicity comes from the fact that in a form the content needs to be passed and then the response will come after a page refresh, but with AJAX the requested information, that is the result of the user's action will come in the actual page. It saves time and gives to the user the impression that the application is really interacting with him.

The Business Logic Layer

The central layer of the model deals with the logic of the program. It receives data from the upper level and transforms it, using in the inner application logics. It also retrieves data from the deepest data level and uses it to the logics. And, by integrating these two processes, it can do modifications in both levels as well.

The Business Logic Layer contains the determinant part of the application logic. It includes:

- performing all required calculations and validations
- managing workflow
    - state management: to keep track of application execution
    - session management: to distinguish among application instances
    - user identification
    - service access: to provide application services in a consistent way
- managing all data access for the presentation layer

The Business Logic Layer is generally implemented inside an Application server (like Microsoft Transaction Server, Oracle Application Server, or IBM WebSphere). The Application server generally automates a number of services like transactions, security, persistence, connection pooling, messaging and name services.

Using the same example of the last session, of the user that wants to see the weather in a specific place, when the information is given by the user the application retrieves it and process. For example, the user wants to see the weather forecast for two days. The application receives its request from the UI and the data is sent to the server. A PHP script catches it and then make the calls for the lower level to get the needed data. When a response comes, being it the desired information or a failed request, it is dealt and then prepared to be sent again to the upper level.

The tools used in this level are usually server-side scripts like PHP,ASP.NET, Ruby or CGI. There is even a solution that uses server-side Javascript, called node.js. These technologies, following the information feeding that comes from the upper level, can do any computational pro- cessing that is needed. The CGI (Common Gateway Interface) scripts are an older technology that defines communication way between a server and the content-generated program, in this context called CGI script. One of the most remembered languages when talking about CGI scripts is Perl. The other languages here cited have a similar approach, by running inside a server. PHP is related to Perl, being as well a scripting language and having similar philosophies. It is one of the most popular languages, being the implementation language of important content management systems as Drupal or Wordpress. Ruby have a large popularity too, especially with the framework Ruby on Rails. Applications as Github or Redmine are built using it. There are many others, of course, and different uses of them, one example being C used as CGI or the Java Server Pages (JSP).

The Data Layer

The deepest level in the layered architecture, the data layer deals with data retrieval from its sources. It is an abstraction to get the plain data, that can be in a wide variety of forms. Once again, it plays a huge role on the reusability and exchange of technologies: if one data source is changed to another, but the proper data is still the same, a good layered design can help by providing the same data to the upper level with the same interfaces, changing only its inner logic.

In the example given in this paper of the weather forecast, the requirement by the user for the next days forecast will come to this level as a request for the forecasts that it may have. Then a search will be made in the data using the given parameters, and then the data (or some information about not getting it) will be sent again to the upper level.

The technologies used in this layer are database management systems like MySQL or PostgreSQL for relational databases, NoSQL management systems like MongoDB or Apache Cassandra, or even plain XML files or text files. For the management systems usually an API will be used for making queries and retrieving data, and for the plain text ones a script will do the needed operations. Inside it there can be any level of sophistication desired by the application designer, so there can be integrity checks, stored procedures, and virtually anything needed to maintain the data in the desired state.

Deepest in the Data Layer: NoSQL and NewSQL

Inside the Data Layer, as it was outlined, many different technologies can be used. Most of the web applications currently active use relational databases, but now the market is seeing a change of paradigm in the form of the NoSQL. NoSQL is a general way to identify non-relational databases. Fowler summarises some common characteristics that NoSQL databases share:

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web estates
- Schemaless

The key points NoSQL supporters use to justify the need for it is that relational databases are not the best solution for any kind of problem, being a problem of its own the uses. They say it is heavy, slow, and non-scalable to use the relational databases, so the use of NoSQL can be a good way to solve these kinds of problems. The use of NoSQL nowadays seems related to startups that use innovating new technology and social web services such as Facebook and Amazon, that have a great amount of data[2] to deal with and have the need to find new ways to use it.

In fact, that is this demand of large data processing. Under the label of big data lies the concept of large quantities of data generated in the last few years and from different sources and in a variety of different formats. The processing of this kind of data leads to a wide range of uses, from healthcare to criminalistics inferences. Of course, new challenges arises with this perspective. The drawbacks come from the nature of

the data - massive, disperse, heterogeneous. This is why NoSQL can be seen as a solution - it thinks about this kind of problem, trying to solve it.

As of 2014, there are four important categories of NoSQL databases:

- key-value stores, that are basically hash tables
- column family stores, which aim is to deal with vast collections of data spread amongst many different machines
- document databases, versioned documents that are collections of other key-value collections
- and graph databases, where the data is presented as a graph, and then it is possible to divide easily into different machines and the queries are more data-specific than the relational ones

A topic that attracts attention when it comes to the issue of scalability and performance of databases is the so-called NewSQL. It is more a way to recognise " the various emerging database products at this particular point in time". The authors writing about it use the term as an identification of vendors that provide SQL databases that are high-performance and scalable, in the market that is also aimed by NoSQL providers. The aims of NewSQL are also related to the Big Data paradigm.[3]

---

[1] Sometimes tier is different from layer. For it, layers mean the logical distinction of the applications integral parts, and tiers mean the physical structures in where the application runs (networks, computers, servers). But in this work the two words are used as synonyms, and with the meaning of layer, not being the objective to discuss physical tiers.

[2] "The increasing amount of data in the web is a problem which has to be considered by successful web pages like the ones of Facebook, Amazon and Google. Besides dealing with tera- and petabytes of data, massive read and write requests have to be responded without any noticeable latency"

HECHT, Robin and JABLONSKI, Stefan. NoSQL Evaluation, A Use Case Oriented Survey.

[3] "The NoSQL projects were developed in response to the failure of existing suppliers to address the performance, scalability and flexibility requirements of large-scale data processing, particularly for Web and cloud computing applications. NewSQL and data-grid products have emerged to meet similar requirements among enterprises, a sector that is now also being targeted by NoSQL vendors."

ASLETT, Matthew. NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases.

Prev Up Next
Part II. Architectures for the Web Home The MVC pattern - useful but not a silver bullet