TECHNICAL UNIVERSITY OF CRETE

THESIS

Thesis Title

Author:
Stefanos Stathatos

Committee:
Associate Professor Vasilis Samoladas (Supervisor)
Associate Professor Minos Garofalakis
Associate Professor Kostas Petrakis

A thesis submitted in fulfillment of the requirements for the degree of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Technical University of Crete

August 3, 2017

Technical University of Crete

Abstract

Technical University of Crete
School of Electrical and Computer Engineering

Electrical and Computer Engineer

Thesis Title

by Stefanos Stathatos

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor. . .

Contents

List of Figures

Chapter 1

Introduction

Chapter 2

Background

This chapter presents some background for the content of this thesis.

2.1 Models and Model Driven Software Development

2.2 Data Access Object

In computer software, a data access object (DAO) [9] is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS or database schema. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or business components. Essentially, the DAO acts as an adapter between the component and the data source.

2.3 Access Control List

An Access Control List (ACL) [8] is a mechanism that implements access control for a system resource by enumerating the system entities that are permitted to access the resource and stating, either implicitly or explicitly, the access modes granted to each entity. A filesystem ACL is a data structure (usually a table) containing entries that specify individual user or group rights to specific system objects such as programs, processes, or files. The privileges or permissions determine specific access rights, such as whether a user can read from, write to, or execute an object. +Role,Permission

2.4 HTTP

The Hypertext Transfer Protocol (HTTP) [2] is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content,

or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body. HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes http and https. URIs and hyperlinks in HTML documents form inter-linked hypertext documents.

2.4.1 HTTP session

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.

2.4.2 Request methods

HTTP defines methods to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. The most common methods which are used in this thesis are presented below.

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations.

POST

The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

PUT

The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.

DELETE

The DELETE method deletes the specified resource.

2.4.3 Status Codes

In HTTP/1.0 and since, the first line of the HTTP response is called the status line and includes a numeric status code (such as "404") and a textual reason phrase (such as "Not Found"). The way the user agent handles the response primarily depends on the code and secondarily on the other response header fields. Custom status codes can be used since, if the user agent encounters a code it does not recognize, it can use the first digit of the code to determine the general class of the response. HTTP status code is primarily divided into five groups for better explanation of request and responses between client and server, and are presented below.

1xx Informational responses

An informational response indicates that the request was received and understood. It is issued on a provisional basis while request processing continues. It alerts the client to wait for a final response. The message consists only of the status line and optional header fields, and is terminated by an empty line.

2xx Success

This class of status codes indicates the action requested by the client was received, understood, accepted, and processed successfully.

3xx Redirection

This class of status code indicates the client must take additional action to complete the request. Many of these status codes are used in URL redirection.[2] A user agent may carry out the additional action with no user interaction only if the method used in the second request is GET or HEAD. A user agent may automatically redirect a request. A user agent should detect and intervene to prevent cyclical redirects.

4xx Client errors

This class of status code is intended for situations in which the client seems to have errored. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents should display any included entity to the user.

5xx Server errors

The server failed to fulfill an apparently valid request. Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has encountered an error or is otherwise incapable of performing the request. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and indicate whether it is a temporary or permanent condition. Likewise, user agents should display any included entity to the user. These response codes are applicable to any request method.

2.5 REST

Representational state transfer (REST) [4] or RESTful web services is a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. The term is intended to evoke an image of how a well-designed Web application behaves: it is a network of Web resources (a virtual state-machine) where the user progresses through the application by selecting links, such as /user/tom, and operations such as GET or DELETE (state transitions), resulting in the next resource (representing the next state of the application) being transferred to the user for their use. There are six guiding constraints that define a RESTful system.These constraints restrict the ways that the server may process and respond to client requests so that, by operating within these constraints, the service gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If a service violates any of the required constraints, it cannot be considered RESTful. The formal REST constraints are represented below.

2.5.1 Client-server architecture

The first constraints added to our hybrid style are those of the client-server architectural style. Separation of concerns is the principle behind the client-server constraints. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains.

2.5.2 Statelessness

The client–server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition.

2.5.3 Cacheability

As on the World Wide Web, clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

2.5.4 Layered system

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.

2.5.5 Uniform interface

The uniform interface constraint is fundamental to the design of any REST service. It simplifies and decouples the architecture, which enables each part to evolve independently.

2.6 Application Programming Interface

In computer programming, an Application Programming Interface (API) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. An API may be for a web-based system, operating system, database system, computer hardware or software library. An API specification can take many forms, but often includes specifications for routines, data structures, object classes, variables or remote calls. API uses are listed below.

2.6.1 Libraries and Frameworks

An API is usually related to a software library. The API describes and prescribes the expected behavior (a specification) while the library is an actual implementation of this set of rules. A single API can have multiple implementations (or none, being abstract) in the form of different libraries that share the same programming interface. The separation of the API from its implementation can allow programs written in one language to use a library written in another.

2.6.2 Operating systems

An API can specify the interface between an application and the operating system. POSIX, for example, specifies a set of common APIs that aim to enable an application written for a POSIX conformant operating system to be compiled for another POSIX conformant operating system.

2.6.3 Remote APIs

Remote APIs allow developers to manipulate remote resources through protocols, specific standards for communication that allow different technologies to work together, regardless of language or platform.

2.6.4 WEB APIs

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API approach is an architectural

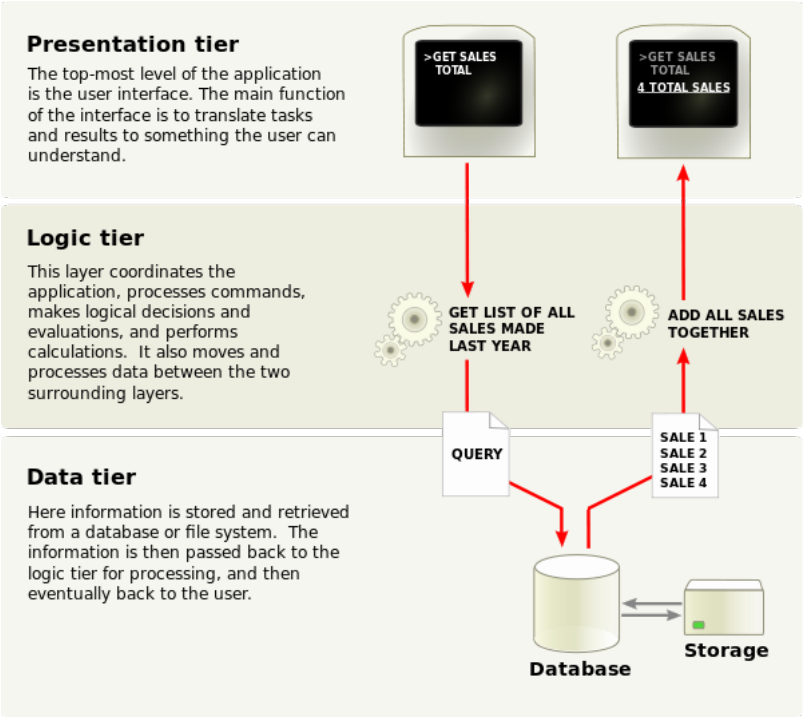

FIGURE 2.1: Three-tier Architecture

approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers.When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

2.7 Three-tier Architecture

Three-tier architecture [6] is a client–server software architecture pattern in which the user interface (presentation), functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms. Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. An image of a three-tier architecture can be seen in figure 2.1. The three layers are presented below.

2.7.1 Presentation tier

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer which users can access directly (such as a web page, or an operating system's GUI).

2.7.2 Logic tier

The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing.

2.7.3 Data tier

The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change.

2.8 Single Page Application

A single-page application (SPA) [5] is a web application or web site that fits on a single web page with the goal of providing a user experience similar to that of a desktop application. In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does control transfer to another page, although the location hash or the HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application. Interaction with the single page application often involves dynamic communication with the web server behind the scenes. There are various techniques available that enable the browser to retain a single page even when the application requires server communication. The most prominent technique currently being used is Ajax. Ajax is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications. With Ajax, Web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows for Web pages, and by extension Web applications, to change content dynamically without the need to reload the entire page. In practice, modern implementations commonly substitute JSON for XML due to the advantages of being native to JavaScript.

2.9 Hierarchical Data Format

Hierarchical Data Format (HDF)) [3] is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data.Many HDF adopters have very large datasets, very fast access requirements, or very complex datasets. Others turn to HDF because it allows them to easily share data across a wide variety of computational platforms using applications written in different programming languages. HDF allows hierarchical data objects to be expressed in a very natural manner, in contrast to the tables of a relational database. Whereas relational databases support tables, HDF supports n-dimensional datasets and each element in the dataset may itself be a complex object. Relational databases offer excellent support for

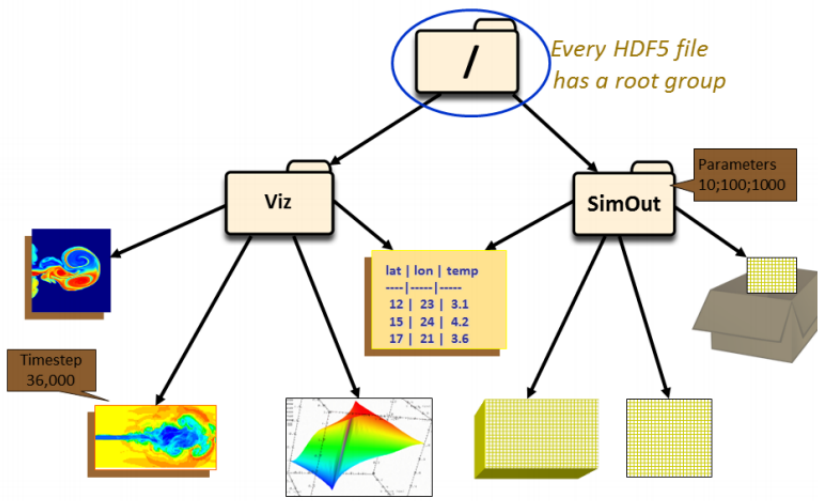

FIGURE 2.2: The contents of an HDF file

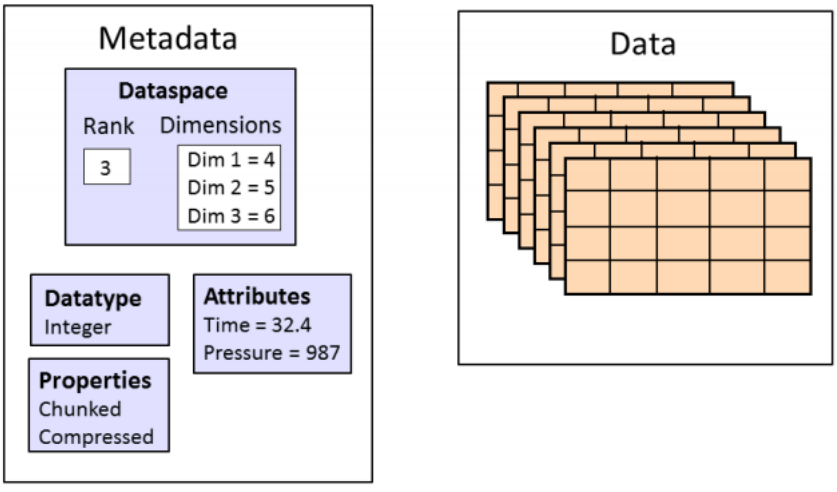

FIGURE 2.3: The general structure of a dataset

queries based on field matching, but are not well-suited for sequentially processing all records in the database or for subsetting the data based on coordinate-style lookup. The contents of an HDF file can be seen in figure 2.2. HDF5 consists of a File Format for storing HDF5 data, a Data Model for logically organizing and accessing HDF5 data from an application, and the Software (libraries, language interfaces, and tools) for working with this format. The data model is described below.

2.9.1 Data Model

The HDF Data Model, also known as the HDF5 Abstract (or Logical) Data Model consists of the building blocks for data organization and specification in HDF5. An HDF5 file (an object in itself) can be thought of as a container (or group) that holds a variety of heterogeneous data objects (or datasets). The datasets can be most anything: images, tables, graphs, or even documents, such as PDF or Excel. The two primary objects in the HDF5 Data Model are described below.

Groups

HDF5 groups (and links) organize data objects. Every HDF5 file contains a root group that cancontain other groups or be linked to objects in other files. Working with groups and group members is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, objects in an HDF5 file are often described by giving their full (or absolute) path names.

Datasets

HDF5 datasets organize and contain the "raw" data values. A dataset consists of metadata that describes the data, in addition to the data itself. Datatypes, dataspaces, properties and (optional) attributes are HDF5 objects that describe a dataset. The datatype describes the individual data elements. The general structure of a dataset can be seen in figure 2.3.

Chapter 3

Related Work and Used Technologies

This chapter presents some background for the content of this thesis.

3.1 Related Work

3.1.1 Plotly

Plotly is a popular public data visualizationcloud service provider. Plotly provides community, professional and enterprise data storage, visualization and analytics services to the user. Excel, CSV and XML data formats are used to upload the data to its cloud servers. Plotly provides online graphing, analytics, and statistics tools for individuals and collaboration, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST.

3.1.2 Loopback

LoopBack is a highly-extensible, open-source Node.js framework which assimilates the best practices of model driven software develpment. LoopBack simplifies and speeds up REST API development. It consists of a library of Node.js modules for connecting web and mobile apps to data sources such as databases and REST APIs, a command line tool, and client-SDKs. A loopback application has three components: models that represent business data and behavior, data sources and connectors, and mobile clients.An application interacts with data sources through the LoopBack model API, available locally within Node, remotely over REST, and via native client APIs for iOS, Android, and HTML5. Using the API, apps can query databases, store data, upload files, send emails, create push notifications, register users, and perform other actions provided by data sources. Loopback is implemented with many of the technologies we use in this thesis. It uses MDSD as a general practice, data access objects for the communication between database and the REST API, access control list for authorization and is written in Node.js.

3.2 Used Technologies

3.2.1 Javascript

JavaScript (JS) [1], is a high-level, dynamic, weakly typed, object-based, multi-paradigm, and interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production. It is used to make webpages interactive and provide online programs, including video games.

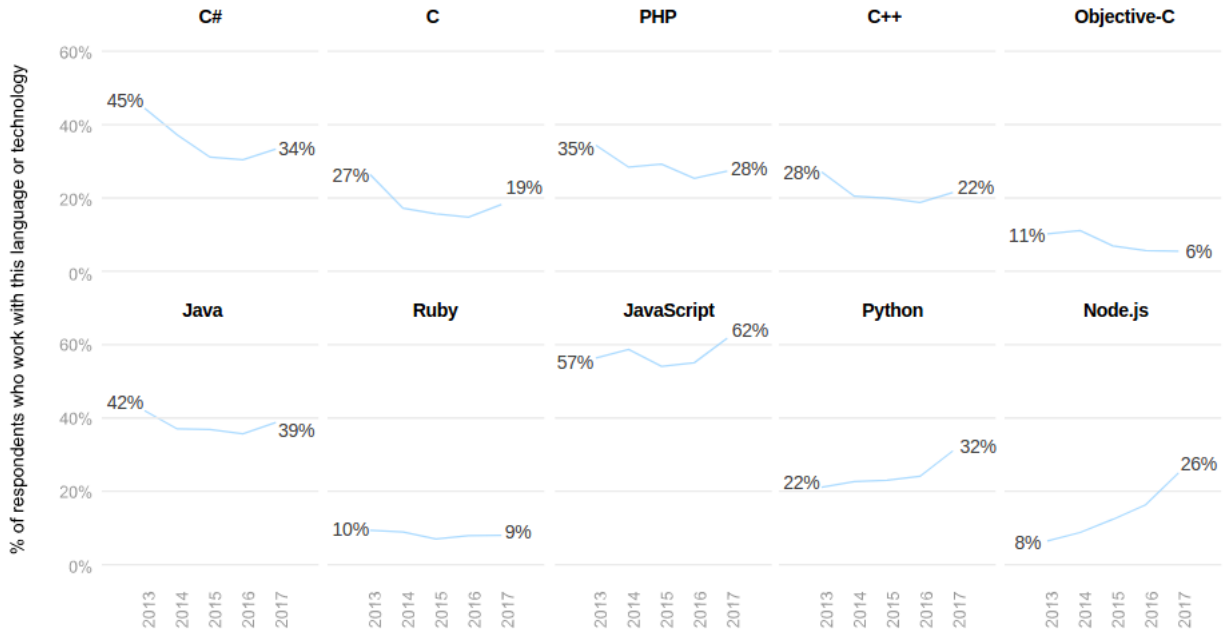

FIGURE 3.1: Developers survey for Stack Overflow website in 2017.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases. The most famous server-side Javascript implementation is Node.js.

Node.js

Node.js [10] is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side.Node.js provides an event-driven architecture and a non-blocking I/O API designed to optimize application's throughput and scalability for real-time Web applications. It uses Google V8 JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript. Node.js contains a built-in library to allow applications to act as a stand-alone Web server. The increasing popularity of Node.js in the last years can be seen in figure 3.1.

Node Package Manager

Npm [7] is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public packages, called the npm registry. The registry is accessed via the client, and the available packages can be browsed and searched via the npm website. Npm is included as a recommended feature in Node.js installer. Npm consists of a command line client that interacts with a remote registry. It allows users to consume and distribute JavaScript modules that are available on the registry. Packages on the registry are in CommonJS format and include a metadata file in JSON format.

3.2.2 NoSQL Database

A NoSQL (originally referring to "non SQL" or "non relational")[1] database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.NoSQL databases are increasingly used in big data and real-time web applications.Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

MongoDB

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas.

Bibliography

[1] Douglas Crockford. *JavaScript: The Good Parts: The Good Parts*. " O'Reilly Media, Inc.", 2008.

[2] Roy Fielding et al. "Hypertext transfer protocol–HTTP/1.1, 1999". In: *RFC2616* (2006).

[3] HDF Group et al. "Hierarchical data format, version 5". In: (2014).

[4] W3C Working Group, W3C Working Group, et al. *Web services architecture*. 2004.

[5] Michael S Mikowski and Josh C Powell. "Single page web applications". In: *B and W* (2013).

[6] Ariel Ortiz Ramirez. "Three-tier architecture". In: *Linux Journal* 2000.75es (2000), p. 7.

[7] IZ Schlueter. "The node package manager and registry". In: *URL: https://www. npmjs. org* ().

[8] Robert W Shirey. "Internet security glossary, version 2". In: (2007).

[9] Inc Sun Microsystems. "Core J2EE Patterns - Data Access Object". In: (2001-2002).

[10] Stefan Tilkov and Steve Vinoski. "Node. js: Using JavaScript to build high-performance network programs". In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83.