Creating Offline Web Applications Using HTML5

Sam Marshall

A thesis presented in partial fulfillment of the requirements for the degree of

Master of Information Sciences
in
Computer Science

Institute of Information and Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
Email: sam.marshall@windowslive.com
Tel: +64 9 414 0800 Fax: +64 9 441 8181

March 1, 2013

Abstract

With the proliferation in the use of mobile devices, browser based applications are becoming the ideal information system for both individuals and organization. Web applications are platform independent and easy to deploy so can be accessed from any device that has a browser. A large number of businesses are now offering cloud services to deliver their software, which are on-demand and pay-as-you-go solutions. The increase in this trend is adding a huge economic and bandwidth challenge to both the network provider and consumer.

While traditional web applications work when they are online, it is however essential for these applications to be available both online and offline modes. With this explosion in the use of mobile devices, the ability of these applications to work offline is especially important in situations where there is intermittent or no network availability.

In this thesis we discuss ways of developing offline web applications. We also propose a method of implementing a wrapper that simplifies the currently proposed W3C's HTML5 client-side database API, IndexedDB, by providing a fluent interface with a Language Integrated Query (LINQ) feel. In cases where synchronization of the client-side data with the server database is a requirement, conflict resolution becomes a bit challenging. We discuss techniques for synchronizing the data that is stored at the client during offline mode with the server database.

Keywords: Web applications, HTML5, Client side storage, Browser databases, IndexedDB, Database replication/synchronization

Acknowledgements

I would like to express my thanks to some people who supported me. First, I thank Prof. Ken Hawick for supervisory, advisory support and for giving me the opportunity to write about this interesting topic.

Thanks also to my wife, who has been there for me and our two beautiful kids, Sean and Chantelle-Rose despite not seeing much of me.

And last but not least I would like to thank my parents and my friends for their ongoing encouragement, and their willingness to understand my research topic, even though they do not have any clue of what I am talking about.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Overview

The use of web applications is increasing lately, to a point where they are now more preferred than desktop applications. Web application are flexible, platform independent and easy to deploy hence address some of the major limitations of conventional desktop applications (1). Cross-platform compatibility allows web applications to reach a wide audience with the least amount of effort. Web applications, unlike their desktop counterparts, do not have to be installed or configured at the user's machine before they are used. This allows users to move from one device to another and still have the same experience (2). This helps users to take advantage of applications without the need to worry about application updates, adding to a better user experience.

While the majority of web browsers have generally been able to cache some of the static resources on the user's device, developers had no way or specifying what they wanted to be cached. This caching technique did not include results from calculations performed by client scripts or storing of dynamic data. The use of cookies comes to the rescue for many developers when dynamic data needs to be stored on the client's computer even though very limited storage space is provided for cookies (3).

Web applications today make extensive use of client scripting languages like

JavaScript (4). Developers are including a lot of client side logic in order to limit the number or requests and responses between the client and the web server. In order to further reduce this traffic, users should be able to use web applications without requiring a network connection.

Nevertheless, developing offline web applications calls for careful design considerations. In this research paper, we look at a number of techniques that can be used to aid in the developing of offline enabled web applications. It is an immense challenge to develop and maintain rich web applications. As traditional web applications work by sending requests and receiving responses from the server, a lot of businesses or organizations are investing a lot of effort and money into making sure that their web servers balance the workload and reduce bandwidth usage (5). The web application and the web server are tightly coupled, and with more requests coming in the web server, the requests take longer to complete due to network congestion and increased web server workload.

Asynchronous technologies like AJAX (Asynchronous JavaScript and XML) have been used to help in maintaining or providing a better web user experience (55). However, building web applications that are responsive can be a problem especially in situations where bandwidth is low or in situations where there might be intermittent network connectivity.

In general, web applications are sand boxed, and do not have access to computer's on-board hardware or other software installed on the computer (6). With the recent increase in mobile devices and online gaming, computer graphics calculations, GPS, camera control are a necessary requirement. When web applications gain access to some of these local resources, it will help them to behave more like traditional desktop applications.

In the web application architecture, both the data and the application reside on a central remote server (7). This is the reason why web application deployment is an effortless task (8) as it means it will only have to be installed or updated in one place; the web server. However, with this solution a network is always required anytime users want to work on the application. The use of these network resources

can be a limitation because of network coverage issues.

This complete network dependence can be affected by a number of issues:

- In cases where devices are reliant on wireless networks, some areas can have limited coverage.
- There might be intermittent network connectivity due a drop in network signal or due to physical obstructions, for instance, tall buildings or underground rail systems.
- Some devices could be connected to slower networks.
- For high network users where there is too much data to be transmitted, bandwidth charges might be high. Some internet providers automatically switch their users to dial up speeds, once they have used up their bandwidth.

A good way to reduce the network load is to reduce the number of requests the browser issues to the web server which will in turn improve web application performance (9). This can be done by storing a copy of the web pages or the resources needed to load the application on the client device.

With recent technological developments, HTML5 provides a solution to this by making web applications to be available in offline mode which allows users to access the application in situations where there is no network or limited network access.

1.2 Problem Definition

The need to support mobile devices and people who work at home or away from their usual workplace is increasing. Therefore, it is vital for a business to make sure that its employees continue work even when they might not be in the normal working environment.

Developing an operating system independent software application that can also work on a variety of devices is challenging. The need for mobile applications is increasing due to the increase in the use of mobile devices. Browser based

applications are becoming the ideal information system for both individuals and organization alike because they are easy to deploy and are platform independent. Most of the browser based applications available at present cannot work in offline mode as they are heavily dependent on a remote database and hence require real-time access to that server. It is however essential for these applications to be available in both online and offline modes (10). The ability of these applications to work offline is especially beneficial in situations where there is intermittent or no network availability.

Figure 1.1 shows a typical 404, "Page not found" on a Firefox browser error page, displayed when a web page requested cannot be found or is not reachable potentially due to network unavailability.

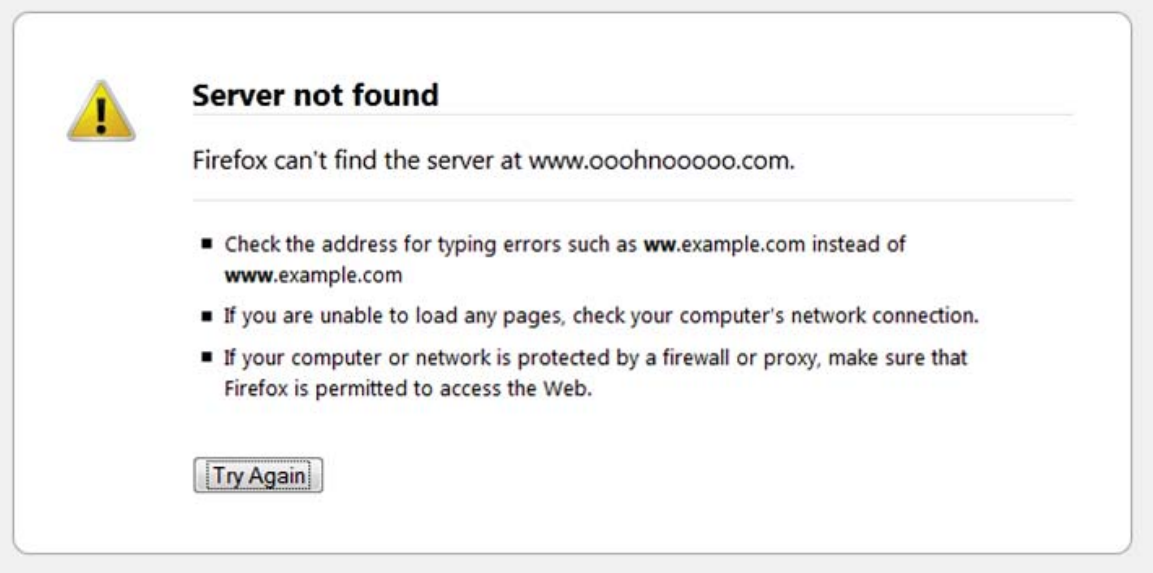

Figure 1.1: Page Not Found

In a traditional Web application scenario, with no network connectivity, no work can be done using the application. The power of desktop applications has not fully been realized in today's web applications as they are difficult to develop due to the nature of their complex software architecture, forcing developers to re-develop, and duplicate web versions of the desktop applications.

The web user experience can considerably be reduced if network connectivity to the web application is not adequately responsive because of low bandwidth and latency.

Below is a list of some of the issues experienced when web applications connect to a remote web server.

1. **Network Requirements**
   In order for users to continue having access to the data stored on a server, it is essential for their devices to maintain a persistent connection with the server. When connecting in areas where there is intermittent network connectivity, this will possibly be a problem and as a result it will affect the user in performing the desired task effectively.

2. **Data Access Speeds**
   In a classic client/server business setting, consumers have fast networks that permit them to have fast access to their data. However, when accessing data remotely, network connections can be over slow wired or wireless links. Whenever the user requests data, it is transferred from the server and this might take some time.

3. **Single Point of Failure**
   When all users are dependent on one server, if that server becomes unavailable for any reason, the entire user base will be cut off from accessing their information (11).

1.3 Objectives

The main objective of developing offline web applications is to improve the user experience by bringing to the web, the power attributed to desktop applications, by offering a web application that feels, looks, performs and works offline like a traditional desktop application.

There are a few problems that offline enabled web application will address due to the traditional web application's dependency on the network (12).

- **Scalability**

  With the need for using web application increasing, it is extremely important for the application to be able to support a large number of users, sessions and also be able to perform a large number of operations.

- **Performance**

  Performance of a web application refers to accomplishing of tasks at an acceptable level inside a given time period. It is however dependent on how the web application is used. For example, a web application that performs financial analysis in money markets need to operate in real-time for all users of the system.

  Online web applications have the potential for poor performance, due to fetching data remotely as a higher number of users send requests to the same server simultaneously. An offline application should be able to make use the local resources and work at an optimum level.

- **Responsiveness**

  Responsiveness is a vital quality-of-service characteristic of a good web applications (13). If the web application's response time is slow, this contributes to a terribly bad user experience. The time it takes the user to get a response should be as minimum as possible, so there is a need to reduce the number of round trips to and from the server by making the application become available on the local device.

- **Fault Tolerance**

  Fault tolerance refers to the capacity of a web application to come back online after a system outage with the least possible interruption. In order to provide a high performance and a highly scalable web application, web application should be designed in such a way that there is limited downtime.

- **Availability**

  Ability to work offline means that, at any given time, there is always a high degree of application availability.

- **Impact on Downtime**

  The impact caused by the server or applications downtime should be minimal.

- **Cost**

  With the recent increase the number of business offering cloud services, which are on-demand, pay-as-you-go solutions, a number of consumers will want to limit the number of times they connect to the Internet (14) as this has an impact on the bandwidth. When an application is working in offline mode, there is no need for the frequent request/response to and from the server as all the application data and processing are on the local device.

The responsiveness and scalability are essential attributes of a good web application. Slow and unresponsive web applications provide a very poor web experience to the users which can lead to decreased productivity and loss of revenue.

For web applications to work in an acceptable offline data mode, user agents will have to be able to store large amounts of data locally with efficient data retrieval and manipulations functions.

The need to have highly portable web application also helps by adding inspirations to the effort put to improve and develop web technologies and tools used to solve these offline issues (15).

1.4 Security And Privacy

The security and privacy of user information on the web should be of concern particularly with the recent widespread adoption and explosion in the use of web applications on mobile platforms. In order to cater for web data security, defences should be put in place to alleviate threats to user data (16). If security and privacy

issues are not addressed, this could have enormous ramifications on both the user and the organization.

Users need to have trust in the web application they work with. The users need to have confidence that the application providers will be able to fulfill their promises and be dependable in their policies and their work ethics and stay in line with the law.

Trust is a measure of the way the web users have confidence in the objectivity and morality of the software providers. This is of significance since it influences the way users and software providers work together and is crucial if businesses, social bodies such as governments, are to work effectively (17).

When designing web applications, information that has personal user data requires protection from unlawful access. Any user's personal information that is collected using the application should only be used for the purposes it was initially intended. (18). Company websites usually have pages that provide privacy policy information, but will need to keep their promises in order to a workable relationship with their intended users.

1.5 Volatility of User data

User data can be volatile as this can change very often and can be easily be lost if due consideration is not taken in designing the user data management system. Any data that is entered through a website or web application can be hugely important to the user, be it a small shopping list or personal bank account details. This data has value to the user and consequences of losing that data should not be under estimated. This thesis looks at some of the circumstances that are vital to think through when designing offline web applications;

Not all user data should be considered as volatile. Generally in web applications, when data is required, a request is sent to the web server to pull the data from the database. While this might seem sensible that the up-to-date data is accessible to the application, the basic notion is that the data the user is working with

is always changing (19).

In the majority of applications, it is vital to study and work out the data that is volatile and the data that is not volatile. If we take a look at an e-commerce website, most of the content e.g. the product catalog can be very static, while the inventory data is less static; The user shopping basket is reasonably unpredictable. However, because of the typical large size of inventory data, it cannot be cached easily. The catalog data can easily be cached because it small and seldom changes. The contents of the shopping basket data can easily be cached as the basket gets updated using a well defined process, which is, by the user adding products to the basket. So in this case the inventory data is the one that required frequent retrieval to make sure that the data is always up-to-date.

1.6 Thesis structure

Developing web applications that work offline poses a lot of challenges (93). This thesis explores the use of offline enabled web architectures and how we can build efficient, fast, responsive, secure offline web applications.

The thesis also looks at the current IndexedDB API specification and the use of a fluent interface in order to simplify the API and provide an interface that makes it easy for developers write simple and more readable code.

When using an offline web applications, data will be stored locally, the thesis then explores the different ways of synchronizing that locally stored data with the remote server. We also look the security and privacy issues that are at the core of offline enabled web applications.

Chapter 2

Literature Review

2.1 Introduction

2.1.1 Web Application

The World Wide Web was initially intended for presenting information and creating a platform for information sharing with everyone around the globe (20). The intention was to create a system that delivered a universally consistent interface to share information. The web content would be coming from all sources around the world. This means the structural design needs to efficiently use a network connection. However, with the advancement of web technologies and growth in business requirements, web applications have developed into complex distributed applications (7). Web applications generally require a network connection to work correctly.

Web applications are increasingly more favored over desktop applications.

- Web applications are easily accessible.
- Web applications only require a web browser to be present at the client computer.
- In both desktop and mobile operating systems, web browsers normally come pre-installed (21).

- It is easier and cheaper to maintain and deploy web applications than desktop applications.
- Web applications do not require any installation on the client's device or computer.
- Application updates only need to be done at the server.

Although the idea of the World Wide Web is geared toward an always online architecture (22), technology is not there yet, as there are always situations where network connectivity cannot be assured. In some situations, it might not be the network to blame but the device itself. In both cases the user will still want to continue with their work. It might be that the user has run out of bandwidth or the user might just want to disconnect from a public internet service in order to limit the risk of their data being hacked.

By 2015, it is thought that mobile users are going to outnumber desktop users. Mobile users now account for over half of all traffic in Asia and Africa. Recent statistics show that, in Africa, more people use mobile phones to access their bank online than they do on an actual terminal (24). This increase has however been attributed to the lack of fixed telecommunication infrastructure in Africa.

The various components of a web application can reside in more than one server sometimes called a web farm. These servers can be running any operating system and different kinds or web server software, but they always have one common thing; the server web pages and the web objects which use the same HTTP protocol to access the web (25). A web application is made up of HTML, CSS, JavaScript and other multimedia resources. In an offline web application the local database interface is coded in JavaScript. Strictly speaking, any language that runs in the browser, for instance, VBScript could be used but in practice, there are no real serious alternatives to JavaScript at the moment. All these resources are requested from the web server and delivered to a web client, typically by a browser running on a computer or mobile device.

The web application interfaces with the client device by virtue of a technology

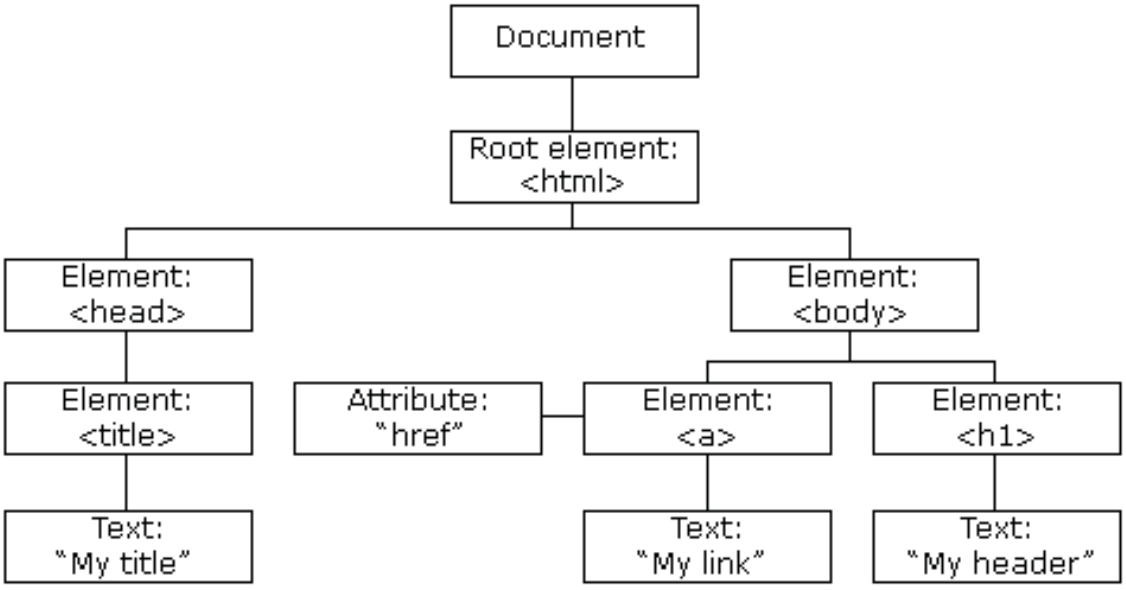

Figure 2.1: HTML DOM Tree (23)

called the Document Object Model commonly known as the DOM (26). The DOM allows the Javascript component to access various features of the client allowing interactive applications to be developed (27). The various features of the client are defined in terms of the DOM. The document object model is the interface point that allows access to local storage in the client device. This is the reason why the database interface is developed in JavaScript. Javascript runs in the browser and has access to the DOM. All the various local storage schemes share this common architecture.

In order to build HTML5 web applications, we need to understand the common architecture that these applications share.

HTML5 offers a new application programming interface (API) that allows us to cache pages even without first visiting the pages. By visiting a single page you can pre-fetch content for other pages. This option requires developers to look at the way the application is used and determining if it is necessary to pre-fetch a particular resource. HTML5 offers an online/offline detection mechanism by functions

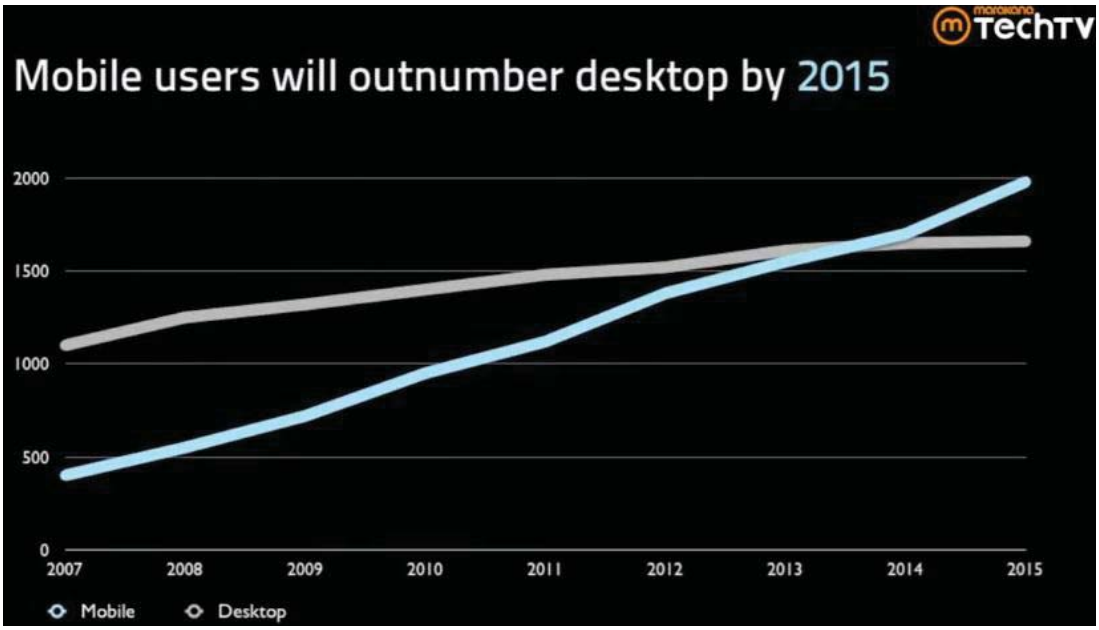

Figure 2.2: Mobile Users versus Desktop Users (28)

that can be called to check if the network is available (29).

2.1.2 Online Web Application

The architecture that surrounds a typical web application is that you need connectivity to the Internet in order for the application to work correctly. The web page is served to the user and the user interacts with the page and submits a request to the server and it takes these round trip too and from the server in order for the application to work correctly. In situations of high latency or no network connectivity, this can be a big problem since the application will have to wait for a response in order to continue, resulting in undesirable, unresponsive or less responsive user interface.

Web applications today are a desirable alternative compared to traditional desktop applications by virtue of them being easy to deploy and upgrade. Web applications are deployed on a central web server and this removes the need

for the application to be deployed on each and every device that requires the application. With this model businesses are able to easily distribute the application to a large number of users.

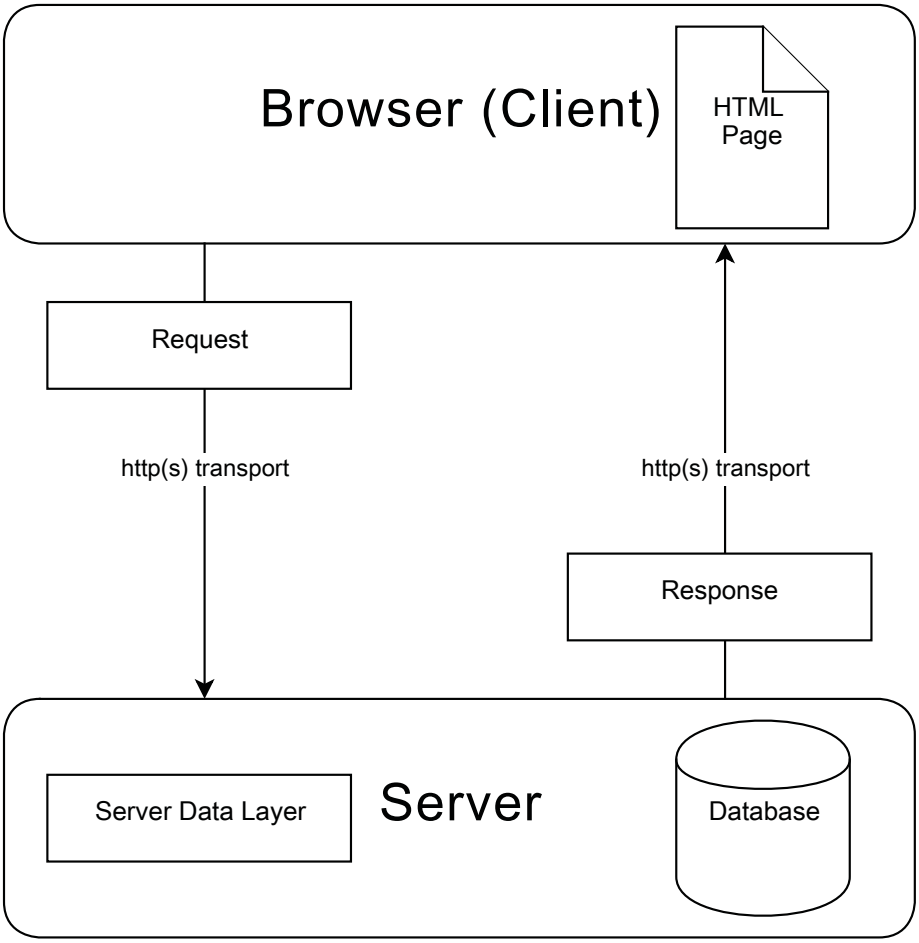

Figure 2.3: Core elements of an online web application

2.1.3 Offline Web Application

The idea behind offline web application is that instead of operating like a standard web application, an offline web application will continue to work even if there is no internet connection.

An offline web application should be able to persistently store both the user

data and the web application resources on the client device rather than fetching data from the server every time a request is made. This allows developers to create rich and responsive web applications as the application will be able to use the locally stored resources. Working offline can help increase the application speed when querying data since the data is stored locally as there is no need of performing the usual round trips to and from the server.

The ability to work in an offline mode is useful mainly in areas where the application is required to work even though there is no network access (30). This is achieved by making sure that the resources and the data that is required by an application or part of it, for it to work correctly are copied to the user's computer or device, in a location where they can reliably be retrieved. Depending on the size of the application, this is normally just a part of the application, because with business web applications, there is usually a database or some web services involved which obviously cannot all be downloaded to the client's computer. The offline application should try and take care of more of the required information but there are certain exclusions within the architecture of an offline web application.

For the offline application to work effectively, it is important to make sure that data stored at the client-side is always valid, secure and up-to-date.

With the recent development in web standards, HTML5 provides an interface for applications to work with local files through the **File API** specification. This means applications can maximize on the power provided by the client computer to perform things like resizing of images on the client computer instead of uploading a huge file and only to resize it at the server (10).

HTML pages

A standard web application is made up of HTML pages that contain the content to be rendered in the browser. HTML page is the main part of the web application as all other resources are loaded from the HTML page.

JavaScript

JavaScript is an interpreted computer programming language (31) that comes as part of the web browsers and used to run client-side programs. JavaScript functions

are loaded as part of the HTML page or as files that get loaded when the browser loads a HTML page. The functions can be handling the page events that occur as the user uses the web application or for manipulating DOM elements

Cascading Style Sheet (CSS)

It is a presentation language that is used to define how an HTML page has to be displayed. JavaScript can also be used to programmatically set styles to DOM elements.

Database

The new HTML5 standard brings local storage APIs. Most of the major browser vendors now already have the API available in their latest web browser versions. The web browser comes with an embedded database called IndexedDB. JavaScript is used in the IndexedDB database to query, insert, update or delete data.

Manifest

In order to use the application cache and decide the files to be used during application offline mode, a manifest file is used. All the required files that are declared in the manifest are loaded when the page that declares the manifest file is first loaded and will continue to be available during offline mode (3). Any subsequent reload of the web page will not attempt to reload these cached files. The pages are only reloaded when the manifest file itself changes. This change in the manifest however causes all the files to be downloaded again even though some of the files might not have changed.

Offline web application work can provide rich, responsive user experience as they take advantage of limited or no round trips to and from the server. In today's business world, with the wide usage of mobile device such as tablets and smart phones which can wonder in and out of network coverage, the capability of these applications is a very important requirement.

Offline web applications should be designed in such a way that when the application is online, it downloads as much information as it needs to be capable of providing as much functionality as it can when working offline. However, the application should be able to synchronize with the server as soon as it goes online in

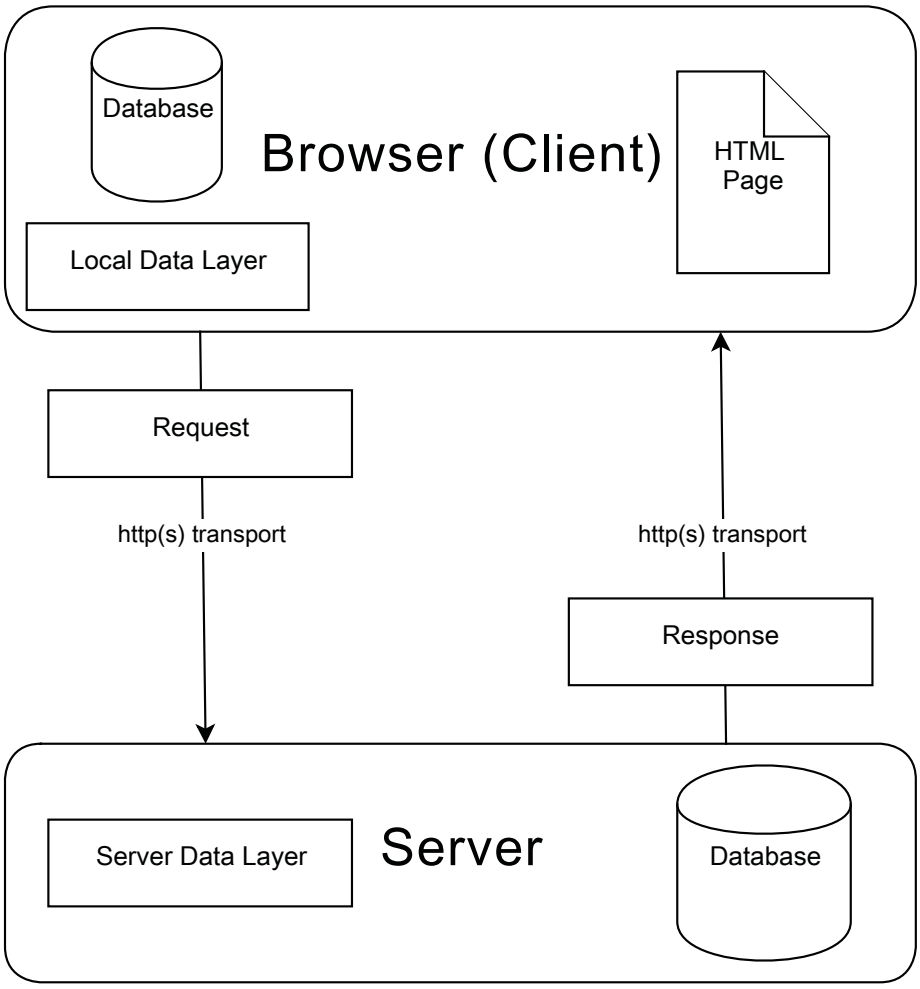

Figure 2.4: Core elements of an offline web application

order to maintain data consistency.

When designing this type of application, there are some significant architectural decisions to be made. When a small part of the application that provides the offline mode is changed, it should not update the whole application. It is important to have a process in place to make sure that data is always up-to-date and should any changes in data occur, then these changes must be reconciled with the server appropriately.

2.1.4 Latency

Latency refers to interruptions or delays that occur when data is transmitted from one point to another. The speed of data transmission over electrical or optical circuits is restricted by the speed at which light travels. The pulses are about two thirds of the speed of light when in wire or fiber transmissions. In theory, transoceanic transmission delays range is around 100-milliseconds (54).

Network congestion is another huge cause of latency. Data packets are put in queues in times of high network usage and this is a substantial source of latency. Data packets are sometimes discarded when the network queue is full, although this can eventually be re-sent using reliable protocols like TCP. Furthermore, throughput can greatly be reduced with any small amount of data loss (54).

2.1.5 Web Caching

When browsing the web or using a web application, speed and efficiency are important. Users are not patient when working with slow websites (32) but at the same time web application providers want to make the most out of their available bandwidth. When web application caching requirements are designed properly it helps reduce network traffic thereby improving the time it takes to access the required information (54).

Caching is an optimization technique that is used to increase performance and scalability of applications. Application Cache provides the means of storing copies of the web application pages including the related CSS and other resources on the local device so that they can be used when the network is not accessible (33). It is important to store the content if the cost of keeping this data is less than that of retrieving it as this stored content may or may not be used again (34).

When a page request is sent to the web server, a number of resources are usually sent back to the client. Any subsequent request of the same page will result in the server sending the same resources over and over again. However, a proxy server or web browser can store the web page content locally and decrease

the bandwidth and time that is needed to serve the whole page again. When a web user requests content from the server, instead of sending a new request, the caching mechanism retrieves the requested content directly from the local cache (35).

Web cache is categorized into two;

1. **Public cache**

   A public cache is used or shared by a number of users. A proxy server is normally used for caching resources that are widely used by a group of users such as employees of a business or customers of a service provider.

2. **Private cache**

   A private cache on the other hand is only used buy one user. This is stored by the browser on the user's machine. This means any resource that is stored on the user's machine will be served instead of loading the resource again from the web server.

2.1.6 Web Application Performance

The performance of a web application can be measured based on how fast it responds to user interactions, its throughput, and how much it uses the resources under a given workload. Responsiveness is measured based in the point-of-view of the web user, while throughput and resource utilization are measured in terms of the whole system, taking into account every single web user (36).

2.1.7 Web Application Scalability

Scalability in principle refers to how a web application is able to use additional load while maintaining the same level of performance by adding additional server resources like CPU, hard disk storage or memory or additional servers in a web farm environment.

2.1.8 Web Performance Rules

- Reduce the number of HTTP requests that are required between the web browser and the web server. A lot of modern web pages require lots of HTTP requests in order to be fully loaded due to the number of style sheets, JavaScript and image files that get used by the page.
- The server should send as little information as possible, by reducing or shrinking the size of the response so that it improves the speed of loading the web page.
- Send the information as infrequently as possible, in such a way that all other subsequent requests to the same page will not require the page to be downloaded again. This will result in a very responsive experience for the user.

2.2 Technology Drivers

2.2.1 Cloud Computing: Software as a Service (SaaS)

The rise of cloud computing is a fundamental change in the computing industry. The use of Software as a service (SaaS) has had a noteworthy increase with the rise of the open source movement, and the emerging of the Internet as a practical stage for business processes. Software as a service is a software distribution method where the software end-users access the hosted application through data centres provided software distributors. Therefore, rather than the software providers asking users to install the software on their devices, they host it in the SaaS model and that they usually maintain. The management of the software rests with the software provider (16).

2.2.2 Mobile Computing

The question that comes to mind is "Why Mobile Web applications?". Mobile phones are very small devices compared to the modern desktops. They have

limited storage and processing capabilities and will be very hard to run huge applications on them. This however, is changing with the increase in technology such that today's mobile device specification is a lot more capable than some of their desktop counterparts ten year ago.

Mobile device native applications need to be approved mobile designers before they can be deployed to the devices for the market. With mobile web applications, there is no approval process required, which means applications can be launched on schedule and be updated as frequently as possible without requesting third party approval.

Mobile web application usage in social networking is on the rise such that mobile applications are not only limited to business application but get used by all kinds of users. Some applications that get used in a non-business environment include games, personal banking and online stores.

Although, mobile devices and related network technologies continue to advance, it is still hard to guarantee data security and network availability.

2.3 Database Synchronization

2.3.1 Introduction

Data synchronization is the process that involves a continuous accession of data consistency between two or more data stores. It is essential for both the user data and application resources to be synchronized. An offline web application can either use data without modifying it or it can also modify the data that gets downloaded from the server, in which case the system needs to have a mechanism in place to push the data to the server and handle any data conflicts that may arise. It is vital to have a change tracking mechanism to be in place to make sure that only that data that has changed gets synchronized with the remote server.

Read-Only Data

In order to improve performance of an a web application and not worry about having a conflict resolution process, data that is only used for reference purposes

should be download to the client. In this case, the client is not supposed to make any additions or changes to data. This type of reference data could be application help text, data used in process validation, or other lookup data. The frequency of change of this data is very low. There should however be a way of identifying whether the data is stale and requires to be refreshed. This could be enabled by say adding a time stamp to the data or having some form of versioning system. As there is no need to keep track of changes and handling of conflicts, this makes the use of read-only data simple.

Transient Data

This is the data that is created at the start of a web application session and gets discarded or reset back to its original state when the session ends. This is in contrast with persistent data where data is stored even at the end of the session.

Data Concurrency

The problem with keeping data at the client is that changes to the data that is at the server can happen before the client-side data is synchronized with the server. A synchronization mechanism which makes sure that the client-side data is always up-to-date with the server database is needed. The mechanism will need to handle any data conflicts in order to maintain data correctness and consistence.

Unordered data

The problem of reconciling unordered data is demonstrated as a way of trying to work out the symmetric difference $S_A + A_B = (SA - S_B) \cup (S_B - S_A)$ between two isolated sets S_A and S_B of b-bit numbers. The answers to this problem are characterized by:

1. **Wholesale transfer**

   In wholesale transmission, the data from a host location is transferred to the client so it can be compared to the client data

2. **Timestamp synchronization**

   In Timestamp synchronization, a timestamp gets added to any alterations made to the data and then all stamped data with a timestamp that is later than the previous data gets transferred.

3. **Mathematical synchronization**

   In this case data get synchronized using a carefully worked-out process where data is treated as mathematical objects.

Ordered data

When synchronizing ordered data, two different values θ_A and θ_B need to be reconciled by assuming that only a fixed number of changes such as inserting characters, removing values, or data modification. Therefore synchronization is the process of lowering edit distance between θ_A and θ_B , down to zero. This approach is used in the file system based synchronizations as data is in some defined order.

2.3.2 Data integrity

In some situations two separate applications or sometimes the same application can run at different times which can produce different answers to the same question. In some situations an application would crush when trying to access a record which did not exist when it should.

A researcher at IMB, Dr. Codd came up with a set of rules base on the set theory for database implementers to follow in order to guarantee data integrity and make it difficult to produce results that are not consistent or for records to seemingly vanish. The set of rules became known as the Codd's rules and this forms the basis of what is now called relational databases. Over the years researchers have added to these rules.

A table is used to store entities, and an entity can in fact be a description of some real world object. Each row in a table represents a single entity, and each column in a row describes attributes of that entity. A table stores one kind of entity.

2.3.3 Database synchronization techniques

Offline web application must be designed in such a way that stale data can be refreshed efficiently without data loss . There are a number of ways to make sure that locally stored content is up-to-date.

1. **Periodic refresh**

   Data stored at the client-side or resources cached by the browser need to be invalidated and updated regularly. The web application's synchronization service can be made to constantly check for any new versions of the data hosted on the remote the web server. Once changes are recognized and acknowledged, it then gets sent down to the client to replace the old data.

2. **Eviction**

   With the use of eviction rules, data can be removed from the local storage without touching the database. Data can be removed from the cache based on least-recently-used and least-frequently-used rules. With the list-recently-used rule, oldest data will be removed from the local cache, while the least-frequently-used rule removes data that does not get used frequently.

2.4 Current Issues

2.4.1 Web architecture challenges

The need to create offline web applications presents a few architectural challenges. Web applications are normally built around a client/server architecture where the server performs most of the work and the client (e.g. browser) is merely there for presentation purposes.

However, in an offline environment, the client needs to take charge of most of that work. This is achieved by adding client-side logic that will be run using JavaScript. Depending on the complexity of the web application in question, this can work well or might be hard to implement. Some applications can be very computationally intensive or require very fast data retrieval and manipulations.

JavaScript is an interpreted language and is not as fast as most compiled languages like C or C++. Google developed a JavaScript Engine called V8 which compiles JavaScript to native machine code. V8 is known to substantially increase JavaScript execution.

One of the major challenges that is encountered when transitioning from traditional desktop applications to offline web applications is that we are developing our applications on a technology landscape that is extremely fragmented and evolving very fast (80).

Not all applications functions can be performed offline for example, a requesting for data from an external web service. Some applications make use of a lot of data that cannot all be stored on the client device and some data might not be suitable to be stored at the client-side.

2.4.2 Functional dependencies

When developing an offline enabled application, it is important to identify units of the system that can work independently without requiring immediate access to the web server or the other parts of the system. The functional module should be designed with all the functionality that is required for it to synchronize with the server once it comes back online. All functional dependencies of the unit need to be identified and coded in such a way that all the processes the user performs will not result in a system failure.

2.4.3 Data dependencies

It is important to make sure that all data that is going to be required by an offline application is identified and supplied. As much as data can be identified for a functional unit, in some cases it might be practical to deliver all the data to the client.

2.5 Existing Solution Options

2.5.1 Dojo Offline Toolkit

Dojo Toolkit is a cross-platform, open source modular JavaScript library that was originally developed by Alex Russell, Dylan Schiemann and David Schontzler among

others. It is aimed at bringing simplicity to the rapid development of Rich Internet Applications. The Dojo toolkit gives developers an easy way to reading and writing of cookies and also provides a secure mechanism for storing data on the local devices through its Dojo Storage abstraction.

2.5.2 Google Gears

Google Gears is an open source software that was first developed by Google and allows online web applications to be used offline. Google Gears is composed of the following:

- A *database* component, that resides on the client device and uses a relational database management system called SQLite.

- A *WorkerPool* component, that allows JavaScript to be run in the background and as a result allow the user interface to continue responding without locking. The WorkerPool works like a group of processes executing in parallel and cannot share the same execution scope and as a result changes made to a variable in a worker has no bearing on the other workers. However, a worker in a WorkerPool communicates with other workers in the same WorkerPool by sending messages using *sendMessage()*.

- A *LocalServer* component, that acts as a local web server by storing static resources such as HTML, CSS, JavaScript, images and multimedia files on the client device.

- A *Desktop* component, that gives a web application access to the local device's resources.

- A *Geolocation* component, that allows web applications to become aware of the geographical location of device (45).

2.5.3 Web SQL Database

When one is used to working with SQL in relational databases, Web SQL database looks very interesting to developers. The issue is that, there is a dependence created upon using the structured query language. There are different SQL flavors from different providers, e.g Oracle's PL/SQL or Microsoft's Transact-SQL. There are a number of major browser vendors who are not kin on supporting Web SQL database.

The code listed below shows how we can use SQL in Javascript to create a database and then create a table and insert data into the table.

```
var db = openDatabase('mydb', '1.0', student database', 2 *1024 * 1024);
db.transaction(function (trans) {
  trans.executeSql('CREATE TABLE IF NOT EXISTS student (id unique, text)');
  trans.executeSql('INSERT INTO student (id, text) VALUES (1, "John Doe")');
});
```

Listing 2.1: Creating a table with Web SQL

After creating the table, data can then be retrieved using SQL select statement and we use a callback function to loop through the results:

```
trans.executeSql('SELECT * FROM student', [], function (trans, students){
  var count = students.rows.length, i;
  for (i = 0; i < count; i++) {
    alert(students.rows.item(i).text);
  }
});
```

Listing 2.2: Retrieving data with SQL select

Web SQL Database is basically SQL database on the client proving functionality to write SQL statements on the client. This W3C specification has been discontinued.

2.6 HTML5

2.6.1 Introduction

HTML is a mark-up language used to define the structure and presentation of content for the Web making it an important piece of the Internet. It has been in continual development since its first inception in 1990 (89). HTML5 is the latest version of HTML. Although this version is currently in draft, all the major browser vendors have already started implementing it in their browser and is currently the W3C Candidate Recommendation. With the recent developments in technology, there has been an increase in the need to improve the language.

HTML5 is increasingly closing the gap between web and desktop applications and will hopefully bridge it entirely. Amid all the recent web technologies, HTML5 is one of those technologies that have be broadly acknowledged. It helps provide a user experience that is dynamic, responsiveness, and seamless usability across the various browsers and devices with many striking features (80).

HTML5 comes with new elements such as HTML user interface controls, local storage support, 2D and 3D graphics, video and audio file support and many other interesting new Application Programming Interfaces (APIs) . HTML5 brings an enormous transformation to the web. HTML5 web applications can be deployed on desktop and mobile devices. The new elements can be used to create highly interactive websites and can give eye-catching effects and improve the performance with little or no customization required.

Besides the some of the features we mentioned above, the interesting power of HTML5 is its ability to support offline web applications that can work when there is no network available. In order to allow web applications to work in a disconnected mode, HTML5 offers these three new APIs, Web storage, Offline storage and indexedDB.

2.6.2 Hardware Acceleration

HTML5 brings a very important performance feature to browser rendering; hardware acceleration. With hardware acceleration, the calculation load that would normally be performed by the central processing unit (CPU) is done by the graphics processing unit (GPU) in the client device's graphics adapter. This greatly increases performance as it reduces the use of resources on they client device. This enhancement adds to the web application, some of the power which is usually attributed to rich desktop applications.

The majority of the major web browsers can now use hardware to accelerate some, or all of the stages in rendering an HTML page. There are a few of steps taken using this process. Taking Internet Explorer 9 for example;

1. **Content Rendering**
   In this initial phase, Internet Explorer 9 uses Windows' Direct2D and DirectWrite subsystems to accelerate content rendering. The acceleration at this stage uses the GPU which improves the presentation performance of the most common HTML elements;

   - Images and video resources are downloaded, decoded and transferred into GPU buffers.
   - Complex page elements, including canvas and SVG are composed into transitional GPU buffers.
   - Simple page elements are drawn directly to the web page buffer.

2. **Page Composition**
   During Page Composition Internet Explorer 9 uses Direct3D, which helps internet explorer to increase performance especially in image-intensive situations. Using acceleration at this stage makes use of one of the GPU's most important features: the capability to draw bitmap images very fast. Furthermore, since the GPU's private memory preserves images, it will be quite first to redraw a page.

Intermediate GPU buffers are prepared with directly-drawn data to form the visible web page.

3. **Desktop Composition**
   Once the web browser has rendered the content and composes pages, the GPU is used to compose the final screen.

2.6.3 Local File Access

HTML5's FileSystem API supports the web application to gain access to the local file system in sand boxed environment (45). This allows the web application to be able to store and retrieve files without gaining access to the computer's main system files. This can be used in situations where bigger binary files are required as they are not suitable to IndexedDB storage or where the web application needs to share data with another application that does not require the browser. Below are some of the use cases where the FileSystem API can be used

1. Persistent uploader.

   - When a large file needs to be uploaded to the server, this can be achieved by copying smaller portions of the file into a reserve area where the portion can be uploaded one at a given time. This can be valuable in conditions where there might be intermittent network connectivity, where the upload can continue from where it left off, instead of starting the upload all over again.

2. Video games or application with high use of media resources.

   - Resource can be zipped and downloaded to the client machine. This allows the application to use the file in the local file system.
   - After downloading a file, the file system API will be able to load the next required resource in the background, making loading of the subsequent game levels faster as no further download is required.

3. Audio/Photo editor that works in offline mode.
    - The data blobs are usually relatively huge, partial file writes are possible.
4. Offline video viewer.
    - Application can download huge files for viewing afterwards.
    - Partial downloads will allow the downloaded portion of the video to be viewed offline.
5. Offline Web Mail Client.
    - Should be able to download attachments and save the file on the local file system for offline viewing.

2.6.4 IndexedDB

In 2009, Oracle proposed development of local browser database, IndexedDB. The IndexedDB is a new API, currently in draft, that provides a capability of client browser data storage and facilitates a platform-neutral data caching mechanism (83). It provides an interface to perform indexed key-value object storage and retrieval and at the same time give an in-order deterministic traversal of keys. B-Tree data structures are normally used to implement this as they are efficient for insertion and deletion of big amount of data.

All data manipulation are performed within a transaction scope. Using a policy to make sure that data stored at the client cannot be read by other applications, IndexedDB follows a same-origin policy where access to data store at the client is limited to only one domain, so cross domain data access is not allowed.

Compared to DOM Storage, IndexedDB API allows large amounts of structured data to be stored at the client and uses indexes to perform high performance searches.

IndexedDB also provides synchronous and asynchronous access APIs. The use of synchronous API is only within Web Workers while the asynchronous API can work both within and without Web Workers.

All IndexedDB processes take place inside the context of a transaction; this guarantees the integrity of database activities between a number of requests, for instance, those initiated by another windows or worker threads. If a transaction finishes normally, modifications made for the duration of the transactions are automatically committed. If a transaction is terminated, all modifications are rolled back.

The Key Model of IndexedDB

- IndexedDB API is not the same as a relational database model; it is an object oriented database. This is a significant and fundamental difference that has a huge bearing in the way the application is designed and built.

- IndexedDB save data as key-value pairs, where the value can have one or more properties. The key can be based on either a key generator or derived from the key path that describes the path to the key value.

- IndexedDB does not make use the Structured Query Language (SQL), but uses queries on an index that produces a cursor, which we can use to iterate across a result set.

- IndexedDB stops applications from reading data with a different origin: applications only have access to data from the same origin (domain / port).

Currently, not all browsers support IndexedDB; The statistics shown in figure 2.5 for Global usage are based on the January, 2013 figures from http://gs.statcounter.com/.

2.6.5 Cookies

A number of websites and web applications use cookies as a means for storing data on the local device. Only 4 KB can be used when storing data using cookies, which consist of strings with unstructured data. This 4KB storage limit can be increased by using more cookies, although this method also has its own constraints,

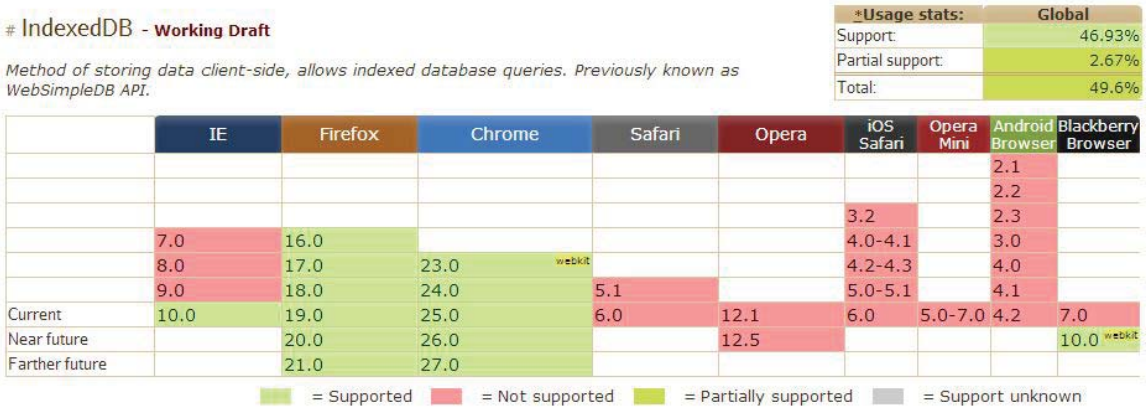

Figure 2.5: IndexedDB Browser Compatibility(37)

as most web browser restrict a domain only to a small number of cookies (38). Web storage either can be window, browser or domain specific.

All cookies set by the website are passed to the back and forth with each HTTP request and therefore the size of HTTP header can also be a restriction. Client scripts, mainly Javascript can be used to access Web storage, while both client scripts and server-side scripts can both be used to access cookies.

Cookies have been in use for a long time particularly in storing HTTPS authentication tokens (40). However, storing data in some form of key-value pairs presents a more elegant approach. When data is stored in key-value pairs, it can easily be manipulated by JavaScript.

2.6.6 Web Storage

Web storage APIs are used to provide local data storage and were introduced in HTML5 as a means of removing the limitations that exist in cookies (41). The data is stored in key-value pairs that is then retrieved by the web applications using scripting languages like JavaScript. The Web Storage API can be used in place of cookies as they provide greater level of security and privacy features. Compared to cookies, it comes with an easy to use syntax to set and get data in JavaScript (44).

Web Storage provides more storage space compared to what is available when using cookies. The data stored using web storage does not need to be part of every request and this as a result reduces the overhead as it does not need to be added the HTTP header. Cookies are transmitted from the client to the server with each request, while with web storage the data is saved or retrieved only when requested, and it's not sent from the client to the server (95).

The are two different flavors to the web storage; local or session storage (39).

Local Storage

This is not a new concept in server side application development. As the user creates a session and navigates from page to page, we are able to persist data across requests. With HTML5 local storage, we now have the ability to do the same session storage in the browser. Local storage is persistent, it allows you to persist data beyond requests. Data will still be accessible even after the web browser or the computer has been restarted (42).

The *localStorage* object is especially useful in situations where data needs to be stored across various windows and persists way after the present session. This object offers persistent storage space for domains. The localStorage stores the data that gets used between web sessions, even when the browser has been closed and reopened.

```
localStorage.setItem('someKey', someValue);
localStorage.getItem('someKey'); // returns someValue
```

Everything in the local storage is restricted. It operates in a sandbox. If the scheme, domain and port do not match, then data cannot be shared in between applications (42) (43).

```
http://example.com:80/
  \       \         \_port
   \       \_domain
    \_scheme
```

Session Storage

sessionStorage is exactly the same as the local storage as far as the API is concerned. The two only differ in that localStorage persists as long as the web browser is open, while opening a new window or tab will start a new session. It is good when working with sensitive data, for example, a banking session, as data will not be persisted when the browser is closed.

Web storage APIs only store strings. Storing data as an object can however be accomplished by using native JSON parsing

```
sessionStorage.setItem('student', JSON.stringify({id: 1, name: 'John Doe'})
    ;);
var person = JOSN.parse(sessionStorage.getItem('student'));
```

Listing 2.3: Storing a JSON Object

2.6.7 Application Cache

In order to provide a good user experience, the majority of web browsers automatically cache web pages visited by the user. This however comes with some restrictions:

- There is no way for web developers to decide on the web page they would like to cache and as a result cannot cache the resources that might be required to have a working web application in cases of network unavailability.
- This kind of caching does not provide a good way to access the cached pages when working offline. In order to take care of the restrictions noted above, HTML5 brings the idea of Offline Web application API (AppCache). With AppCache, developers can decide the resources they would want to be available when the application is offline. This means the developer can decide on what the website till look like when running in offline mode.

The HTML5 application cache, which is often called the AppCache, is a feature geared towards supporting offline web applications. When correctly imple-

mented, the web site will continue to run even in situations of network unavailability. When a single page is requested, all files specified in the AppCache's manifest file are downloaded and any subsequent requests will continue to use the downloaded files whether the network is accessible on not (44).

2.6.8 Web Workers

With the need for rich web applications increasing, the client-side processing is getting heavier. What HTML5 web workers brings is threading to the client browser . This is useful in situations where you have for instance, intensive processing at the client (45), which can often have the user interface becoming unresponsive. Web workers run autonomously in the background, free from any other user interface scripts that could have been started from the same page. This API gives us the ability to delegate processor intensive jobs through a web worker with runs in the background. There is a messaging API that goes back and forth between the browser and the worker to allow the browser to be responsive.

There are two types of web workers; they are dedicated and shared workers.

1. A **dedicated worker** is linked to the browser window that started the worker and has a very tight relationship with that window.

2. However, a **shared worker** runs in the background and basically all scripts that run in that domain can send a script to that worker

When working with computationally intensive processes, web workers enable the user to continue working with the page being responsive (46). Web workers use some messaging system to communicate with the client browser. Web workers have access to the navigator object, which provides access to *appName, appVersion, platform*, and *userAgent*. We also have the ability to use the timing mechanism, such as *setInterval, setTimout, clearInterval* and *clearTimeout*. So if we have to do some sort of delay processing, it will work well with this API. You can also perform AJAX calls by calling the *XmlHttpRequest* object. However, a web

worker does not have direct access to the DOM (10), so it cannot read or update page contents directly.

Usage examples

- Pre-fetching and/or caching data for later use
- Real-time formatting of text
- Checking of spellings
- Background processing or polling of web services
- Processing huge arrays or JSON objects
- When doing image filtering in HTML5 <*canvas*>tag
- Updating many rows of a local web database

2.6.9 Long polling

Long polling is a just another polling method that allows data to be sent from a web server to a client browser (48). With this technique, the browser sends a request to the web server but the server does not send a response immediately back to the client if it does not have the requested data, but waits until the information becomes available or when a defined timeout has been reached or when the client has disconnected.

2.6.10 Web Sockets

Traditionally HTTP communication streams are controlled by the client, the way they work is: the client can frequently request data from the server and the server acknowledges the request from the client and sends a reply back to the client. However, the HTML5 Web Sockets API provides a fully bi-directional (49) socket channel that facilitates communication between the client and the server which will in turn

send an answer back to the client without the client sending a request. This can be useful in situations like financial stock tickers, chat clients, medical device readings, where the server responses could be out-of-date by time the browser page is rendered. In order for the user to get the most recent data, they will have to frequently, manually refresh their browser which will not provide a great user experience.

```
GET /text HTTP/1.1\r\n
Upgrade: WebSocket\r\n
Connection: Upgrade\r\n
Host: www.websocket.org\r\n
…\r\n

HTTP/1.1 101 WebSocket Protocol Handshake\r\n
Upgrade: WebSocket\r\n
Connection: Upgrade\r\n
…\r\n
```

Figure 2.6: Typical WebSocket handshake: web client request and web server response (50)

When we use JavaScript to send requests to a remote web server using the HTTP protocol, the message is constrained by the basic stateless nature of the HTTP protocol (79). However, some requests often require long running connections and two way communication over TCP sockets. It is not safe to provide the client with to low-level TCP sockets. The WebSocket API provides a safe substitute. It provides a way for the client to create two way socket-type messages to the web server through the WebSocket protocol making it safer to make long running bidirectional connections.

Web Sockets address two most important issues in offline web applications today:

1. Network Throughput (Overhead of HTTP)
2. Low Latency

In normal HTTP architecture, the client sends a request to the server which in turn send a response back to the client and this is done continuously. There is a lot of

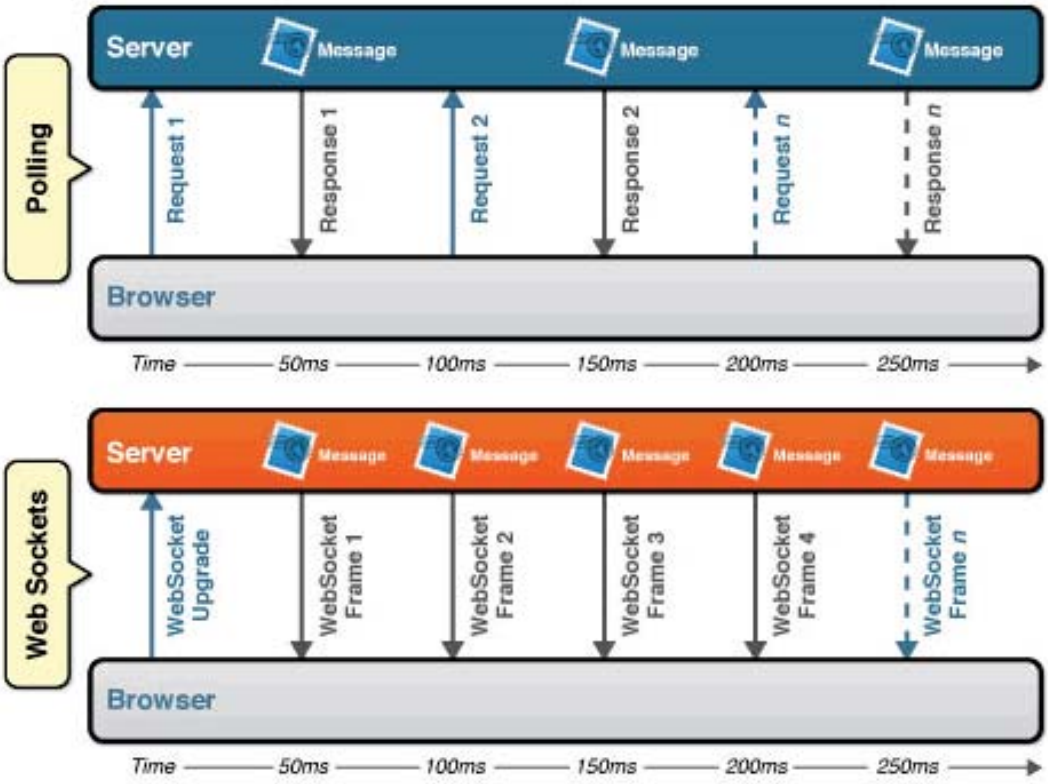

Figure 2.7: Network Throughput: Polling vs WebSocket (78)

overhead with this architecture because of the HTTP headers or meta data going back and forth, consume bandwidth and server resources. This has a huge bearing not only on the performance of the machines but also on how real-time data can be used, because as soon as you make a request, by the time you get the response, data might have changed by then. The way sockets work is that a client sends a request to the server and gets a response and the connection remains established between that client and the server so that the server can continuously send responses to the client without having to require subsequent client requests (51). This results in reduced bandwidth usage and server resources. It gives an opportunity such that any time a change happens on the server, then client can immediately be updated.

If you have too many connections, the network can get congested especially

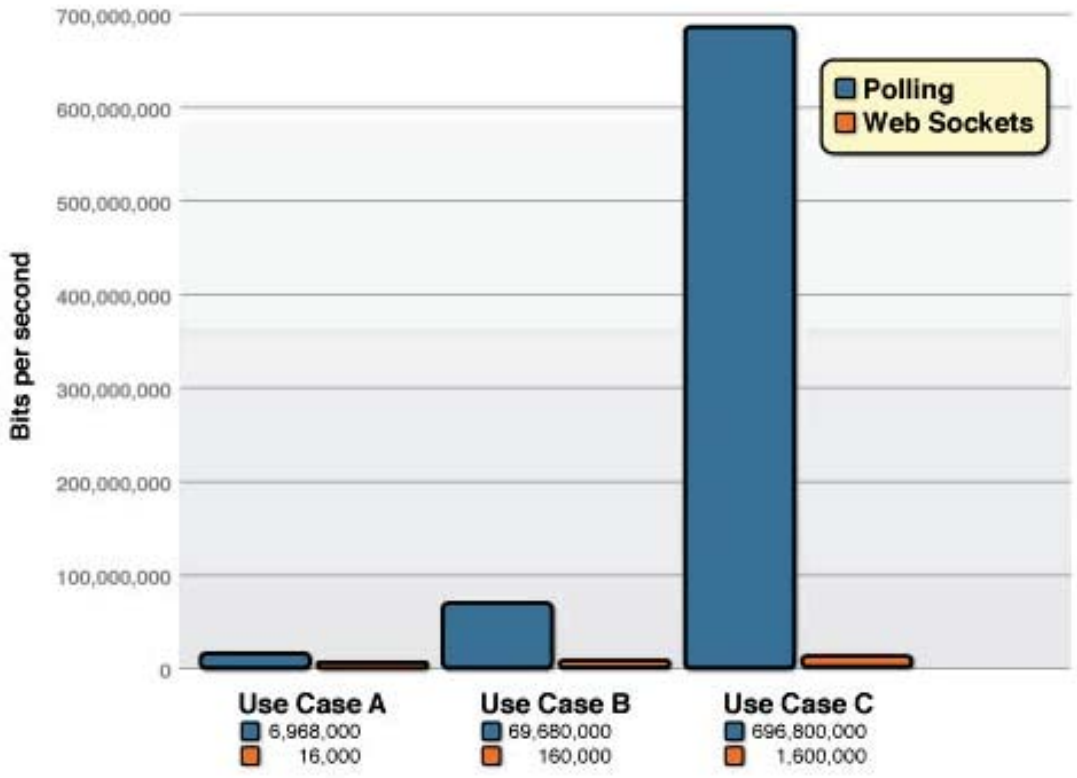

Figure 2.8: Latency: Polling vs WebSocket(78)

if you are dealing with mobile devices or unreliable networks and a server can only support a number of finite connections.

2.6.11 XMLHttpRequest 2

The XMLHttpRequest API is used in web client languages like JavaScript for sending and receiving HTTP or HTTPS data between the web browser and the server. The response can be loaded directly into the DOM and change the page content without requiring a manual page refresh (52). JavaScript can be used to check the response message and manipulated to any required client-side object that might be required. The name of the XMLHttpRequest API is a bit misleading because beside XML, the API also supports JSON, HTML and plain text.

This API is mainly used with AJAX to create responsive web applications (52). It also adds the ability to set timeouts on requests and allows the progress of data transfers to be monitored. Requests that are made using the XMLHttpRequest API support the same origin policy, which makes sure that applications cannot request content from a different domain. However, cross-origin requests are also supported and safe.

Chapter 3

Methodology

3.1 Introduction

Essential requirements of an offline web application

In order to have a successful offline web application, there are three basic requirements:

1. **Locally Stored resources**
   The main requirement is to make sure that the web application can work without a network connection. All static application resources that constitute a web a page like HTML, CSS, JavaScript and image files have to be stored on the client's device in order for them to be available without having to connect to the remote web server. When the network becomes available, only the pages that will have changed will need to be retrieved from the server and stored locally.

2. **Locally stored user data**
   Web applications normally do not just have static pages but have dynamic pages that are created using data normally stored on a remote database. Therefore, this data needs to be available locally in order for the web application to work when offline. Any of the changes that occur in the locally stored data will need to be synchronized with the remote server once the network

becomes available.

3. **Increasing performance**

   Retrieving pages and building dynamic content by accessing a remotely hosted database can be an expensive process that can have a huge effect on user experience as it can block the UI causing the browser to become unresponsive. For that reason, there is a need to build a functionality that makes sure those operations that take time to complete are performed using an approach that does not slow down the rendering or user interface.

In order to make sure a web application works offline, the general implementation is as follows:

- Check network accessibility and enable online or offline mode depending on network availability.

  To check if the application is online or offline we use *window.navigator.onLine*

```
if (navigator.onLine) {
  alert('application is online')
} else {
  alert('application is offline');
}
```

Listing 3.1: How to check online status

We can subscribe to a window event so as to get be notified when there is a change in the network state.

```
window.addEventListener("offline", function(e) {
  alert("application is offline");
}, false);

window.addEventListener("online", function(e) {
  alert("application is online");
}, false)
```

Listing 3.2: Subscribing to window event

- Cache all the data required for the application to work in offline mode.
- Synchronize the state of the client application and its related data with the server when the network becomes available.

3.1.1 Important Design Aspects

- **Hit Determination**
  We need to assess the information to determine the data that requires caching. There are numerous techniques of checking to see the number of visitors or hits for a website or web application. The amount of hits a website has is an essential element in determining the effectiveness of the website. It gives an idea of the number of people who visit the website, including how many times a particular page is visited. It can also point us in the right direction in figuring out the data is normally requested by the users. This can be accomplished by using some of the available tools.
- **Refresh:**
  Locally stored data can become stale. We need to make sure that the cached data is up to date.
- **Eviction:**
  Determine what data needs to be removed in order to create enough space for new data.
- **Coherency:**
  Determine the best way the local storage will merge data. Data updates always need to be sent ahead of queries (47).

3.2 Designing for Performance and UI Responsiveness

Performance is a significant issue for any application, but develops more concern if the application is a web application where a lot of people can access it at the

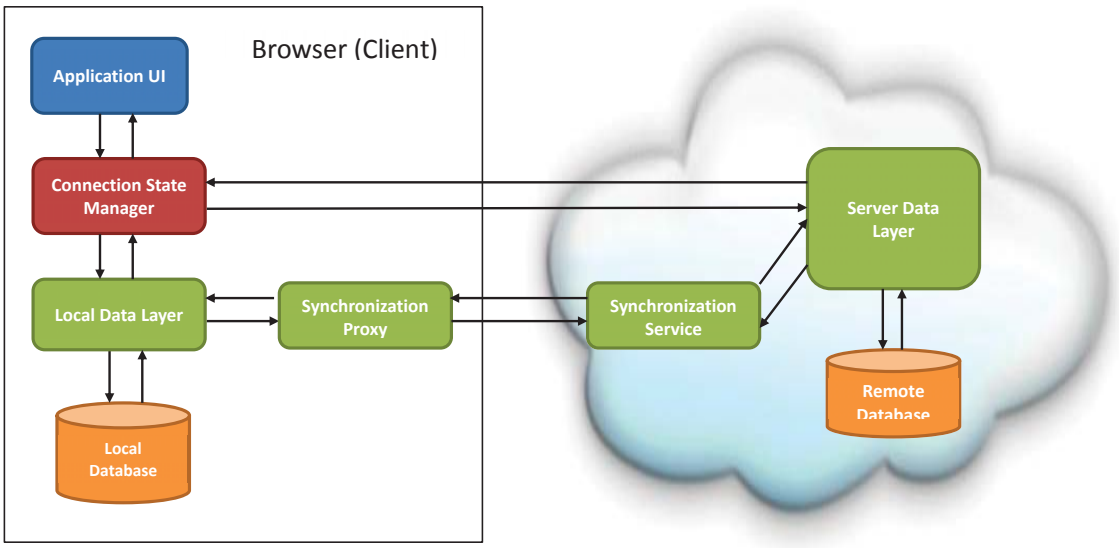

Figure 3.1: Extended: Core elements of an offline web application

same time.

Desktop applications are generally faster (53) and more responsive than traditional web applications. This is because a large portion of the applications resides on the client machine and hence makes most of the local resources. In order to increase the performance of a web application, the application needs to run on a very fast Internet speed. This alone will not however fix the performance problem. An interesting scenario will be to reduce the amount of page requests to the server by making the application use local resources more like a desktop applications. When a portion of the application and data reside at the users's devices, then the application will make limited round trips to the server (81).

There are a number of things that can be examined when designing an application for performance.

- **Data caching**

  The caching of data can greatly improve applications performance as it allows work with local data as opposed to constantly having to request data from a remote server. Caching can significantly reduce loading time espe-

cially in subsequent page request (54). Storing data at the client will obviously require some investigation to make sure that the data at the client-side is not sensitive or frequently changing data.

Caching can significantly reduce loading time thereby increasing performance, particularly in subsequent page requests, if some specific data is read frequently from a remote data repository. There are some data caching design rules to be carefully considered:

- Cached data should be not stored for a prolonged period if it will change frequently.
- Looking at data that is frequently used but unlikely to change could be a good start.
- The server should be able to support notification events if the remote data is changed.

One way of improving performance is to have the application run on a web farm made up of a number of web servers. Having multiple servers is important because the amount of work is shared among the multiple servers and if one of these servers goes down either through a fault on the server or due to network unavailability then other servers will pick up the load. If the server is running on ASP.NET then ADO.NET DataSet objects (55) can be used to cache content that gets accessed regularly, decreasing round-trips to the remote data repository.

There are nevertheless, some likely circumstances when cached data can expire and application might end up using inconsistent data.

- **Content Delivery Network (CDN)**

  CDN is all about storing frequently accessed data closer to those who are accessing it and as a result we have increased availability and performance (56). If you want a web application or website to have really fast access for example, in countries around the world, then copies of it can be stored in

different locations around the world (57). When a web user requests a page that has some of the resources hosted on a CDN network, the request will go to the CDN server and first look in that edge server cache and see of the file is already stored there at which the server will serve the file. This can boost performance because the requests and responses do not have to traverse the entire internet to get to the origin web server. Static files like CSS, images and JavaScript are the most common files to be hosted on a CDN network.

- **Optimizing Network Communication**

  The web application should be designed to such a degree that it does not go through chatty interfaces that perform a lot of request and response operations when communicating with the remote web services, when all it needs is one logical operation.

- **Use threads efficiently**

  We need to efficiently design the web application to avoid blocking Input/Output (I/O) bound calls as the user interface can become unresponsive.

- **Using Transaction Effectively**

  We need to use atomic transactions to make sure that the data stored at the client is consistent with the one at the remote server (58)(59).

- **Optimizing Application Start Time**

  The user will normally get a favorable perception of the web applications performance if the application has a fast startup time. Having the knowledge of how the application gets used by certain users is important, so that the application can reduce the amount of data to be transported by only loading the most likely required area.

- **Efficient Resource Management**

  Designing the application efficiently means the resources are also managed efficiently. Create only objects that are required and dispose those that are no longer required (60). A complete page refresh might not be necessary

and should be avoided if only a small part of the page will be affected when the page is reloaded.

- **HTTP Compression**

  One of the basic performance rules is to send as little information as possible. One effective technique is to utilize HTTP compression (60)(61). A client HTTP request includes an *Accept-Encoding* header, which is used by the client user agent to specify to the server whether or not it can handle compression. With compression enabled, the server will compress the requested resource, making it a lot smaller before sending the resource back to the client and as a result reducing the bandwidth used.

- **Content Expiration**

  Sending content to the client as infrequently as possible is one of the main performance rules. This can be achieved by leveraging content expiration. An expiration date and time can be set for all files in a folder or an individual resource. The expiration dates can be based on how often as resource changes. The files can be renamed by the server if there a is need to force another download before the expiry date (62).

- **Pre-fetching resources**

  Image, video or audio files can be very huge resources and take longer to download than other smaller resources, but users want to view the page as soon possible. One of the techniques that can be used is pre-fetching where a resource is automatically downloaded to the web browser before the user even makes a request to get it (64). In this case, the sooner the request to get the resource is made the sooner it will get to the client.

  This method can greatly increase site responsiveness and hence help with better user experience. This should however be used with care as in some cases the user might not actually require the pre-fetched resource and might just jam the network for no good reason.

3.3 Javascript Libraries

3.3.1 jQuery

Big industry players like Microsoft have now adopted the jQuery JavaScript Library and are working with the jQuery team to speed up the development of new features that help make it quicker and simpler to develop rich web applications. Microsoft is now adding jQuery to its Visual Studio platform so it can be used with ASP.NET web development.

3.3.2 Knockout.js

Knockout is a standalone, open source library that helps developers to create rich, responsive user interface. It is based on the Model View ViewModel (MVVM) pattern and provides a model that helps the user interface to be updated dynamically depending on the user's actions or changes in an external data source.

Knockout provides a smart way of tracking dependencies so that the user interface can automatically be updated every time the data model changes. This is achieved by using adding declaration to HTML tags that associates DOM element with the data model. It can easily be extended by adding behaviors as new declarative bindings that can be reused without writing too much code. Automatic dependency tracking, declarative bindings and templating are Knockout's three fundamental features.

3.3.3 Model View ViewModel Design Pattern

Model View ViewModel (MVVM) is a design pattern for creating user interfaces where the view and the model are loosely coupled. It defines how we can have a potentially complicated UI simple by breaking it into three parts:

- **Model**

  The core focus of the model is that it is the data layer. It is the object graph that provides data for consumption by the view and the view model (65). and

can also contain rules that affect the data such as computation business rules that populate properties. It can also include service calls, persistence logic, data access layer, but the primary focus of the model in the MVVM pattern is that the model contains the data and rules that affect the data. When using Knockout, the view model can communicate with the model by making Ajax calls to server-side logic to read and write the stored model data.

- **View Model**

  View Model contains several things that contribute to its role. First and foremost is that the view model offers data to the view so that the view can present it to the user. It also contains some model manipulation logic which can be to retrieve data and to know when to push modified data to the view (65). There is also interaction logic; this is not core business logic. In a properly layered architecture, business logic should be in its own discrete layer and should sit behind a service boundary so it can be called by multiple or different client applications. Interaction logic handles discrete workflow that happens in the user interfaces that respond to the user actions and is the core logic that decides on when to call the service logic or present something on the client side. When using Knockout, view models are just JavaScript objects that do not know anything about the view (HTML). Having view model abstraction makes it easy to manage complex behaviors.

- **View**

  View represents the structural definition for the user interface or the screen and it's the one that represents that state of the view model (65). It can be both static and dynamic. Static would be the overall layout of the elements you see within the screen while dynamic means that part of the structural definition can include visual state transition and animation that move things around the screen.

Data Binding

A data binding functionality is one of the most powerful features that need to be

considered when developing rich web applications. It needs to be a very loosely coupled mechanism, where the view itself and the elements within it do not have to know where the data source objects come from nor what the data source objects' type is. The fundamental capability of data binding is to be able to pull data into the view and specifically put data source properties into the view elements to set element properties (66).

However, data binding can go into the other direction as well. You can update the underlying data source objects from the view based on user interaction. For Example, if a text box is of some property in the data source and the user types in a new value for the property on that data source object, that data can be automatically saved back into the underlying data source object.

The other way data binding can be used is to tie the different elements of the view. For example, we have a button that will raise a command that the button element will carry the information regarding a piece of context of the command that is executing.

Property/Collection Change Notifications

The general idea here is data bindings need to know when data has changed (65). You might have code that goes and modifies properties on the data source object based on the interaction with the user or you might have something like an asynchronous service call in the background that goes and retrieves updated data and pushes it into your data source objects. So basically what is required is for the binding to re-retrieves the value at that point and assign it to the target property to have that view and the underlying data objects to stay in sync.

3.4 HTML5

Creating web applications that deliver rich features needs a good appreciation of the main Web standards for example, HTML5, CSS3, JavaScript and frameworks that are coming up today. Also, JavaScript has a number of libraries like jQuery, KnockoutJs, PaperJs that can be used to improve developer effectiveness and

reduce web application development with a high quality end product.

HTML5 and jQuery have become broadly recognized that they are becoming the standard when designing web applications that provide rich responsive user interface that works seamless across all major web browsers and device.

3.5 IndexedDB

The IndexedDB is a new API that provides a capability of client browser data storage. It provides an interface to perform indexed key-value object storage, retrieval and at the same time give an in-order deterministic traversal of keys. B-Tree data structure is normally used to implement this as they are efficient for insertion and deletion of big amount of data.

All data manipulation is performed within a transaction scope. Using a policy to make sure that data stored at the client cannot be read by other applications, IndexedDB follows a same-origin policy where access to data store at the client is limited to one domain, so cross domain data access is not allowed.

IndexedDB is an object-oriented database (67) and the data is stored in object stores. However, the IndexedDB API does not provide any way of managing data relationships between two different object stores. All the data relations have to be managed by the application that uses the database. As data is stored in the object stores that are not tied to any other object store, this means that a single object can be requested from the server without the need to bring with it any of the potentially related data.

Despite the fact this approach of retrieving data can be extremely efficient, it is however characterized by a problem of data consistency which has also been generally related to object-oriented databases.

The IndexedDB API is made up of a number of objects, every one designed for particular job:

- *Object stores* are groups of JavaScript objects whose properties have individual values.

- an object, in an object store has a common property referred to as a key path which stores a key value or simply a key. A key is distinctively used to reference a single record in an object store. An index categorizes objects based on the value of a common property.

- Indexes use groups of keys in retrieving distinct items from the object store. A cursor defines a set of values. When an index creates a cursor, the cursor stands for the set of key returned by the index.

- A cursor represents a set of values. When an index defines a cursor, the cursor represents the set of key values returned by the index.

- A keyRange describes a collection of index values or a group of objects in an object store. The collection of keys can be used to filter cursor results.

- A database manages transactions and comprises of various object stores and indexes.

- A request characterizes discrete actions taken against objects in a database. For instance, opening a database results in a request object, and we can create event handlers on that request object to respond to the results of the request.

- A transaction takes care of the context of processes and is used to maintain the reliability of database activities. For instance, we can define an object store basing on the perspective of a version change transaction. If a transaction is terminated, every job in the transaction is annulled.

3.5.1 ACID Properties of Transactions

A database transaction refers to an isolated logical unit of work performed on the data. Transactions form the basis of database integrity and have the following ACID properties:

- **Atomic**

  All work on persistent data should either succeed completely or fail completely. Any data updates makes during with a transaction should all be completed. If there happens to be to be an error or any part of the transaction fails to, the database should go back or rollback to the state prior to the transaction.

- **Consistent**

  We are responsible for defining consistent transaction boundaries. The transaction ensures that the all the system data is in a valid state when the transaction is completed or rolled back.

- **Isolated**

  No changes to persistent data are visible until the transaction commits. By running in isolation, it gives the system the impression that it is the only has function running on the system and make sure that it does not have access to data from another transaction.

- **Durable**

  The server ensures that all committed data is safe on disk and recoverable in cases there is a system failure.

3.5.2 Asynchronous

The running of long, resource intensive client side scripts using JavaScript is normally associated with bad user experience due to the fact that when the script runs for too long the user interface becomes unresponsive. IndexedDB solves this problem by providing the asynchronous API (79). When an asynchronous call is made to the server, it instantly returns with an instance of IDBRequest which does not have any data about the outcome of the process. A callback function gets passed with the call to a function and when the operation finishes and the result is available, a DOM event will be fired and the IDBRequest instance properties will be populated

with the data. The event type can be use to determine whether the operation was successful or not.

3.5.3 Synchronous

With IndexedDB synchronous API, all calls are returned immediately and hence have a blocking access effect to IndexedDB databases. This synchronous API methods are used with web workers because they obstruct the calling thread.

3.6 Application Cache

The application cache can be used to save HTML, JavaScript, CSS, and media files on user devices. This is useful when developing offline web applications because the pages can still be accessed in situations where network is not available.

Application performance can greatly be improved when static HTML, JavaScript, CSS, and other media resources that are loaded using locally stored versions, instead of being retrieved from the server every time and a request is made.

Creating a simple offline application

A manifest file will need to be declared, which basically has a list of names if all the resources that need to be stored on the client machine.

A manifest is declared as follows:

```
<html manifest=‘‘path/filename’’>
```

When a web application initially loads in the client browser the page that declares the manifest is loaded including all the files in the list. All the resources are saved to the local device and will be served in all the succeeding requests. The resources are only downloaded when there is a change on the manifest file. A change in the actual resource that is on the manifest list or a change on the file that declares the manifest does not cause the file to be downloaded; it is only triggered by a change in the manifest file.

The manifest file requires a MIME type of text/cache-manifest and name of the file needs to have a .manifest extension. It uses the domain and scheme policy so

that it cannot be loaded from a different domain. The text CACHE MANIFEST must be in this first line of the page and not even a comment can come before the CACHE MANIFEST.

```
CACHE MANIFEST
# 2012-10-18:v1

# Explicitly cached 'master entries'.
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Files that get used when network is available
NETWORK:
login.aspx
http://api.example.com
# static.html will be used if main.aspx is not available
# offline.jpg will be used as a placeholder for images in images/large/
# offline.html will be used in place of all other .html files
FALLBACK:
/main.aspx /static.html
images/large/ images/offline.jpg
*.html /offline.html
```

Listing 3.3: Example of a Manifest File

The manifest file can potentially have 3 different sections: CACHE, NETWORK, and FALLBACK.

CACHE:

The CACHE section is the default section for all entries and all resources that are added to this section will be cached the first time the files that declared the manifest file is requested.

NETWORK:

Resources specified in this section are network dependent. These will get loaded

directly from the web server when the web application has can access the network. This can be useful as a security measure in situations where we want users to use resources from a secure remote web server instead of caching this resource locally. It is important however to have a fall back measure to keep the application functional in cases of network unavailability.

FALLBACK:

This section is useful as it acts as a way of ensuring that resources that fail either because of problems with the web server or with network connectivity issues can be replaced by alternate locally stored versions of the resource.

All cached pages will always be served from the cache even when the network is available. The cached files only get updated when the manifest itself is modified.

3.6.1 Problems with App Cache

When files listed in the `FALLBACK` section cannot be downloaded due to network failure, the specified alternate file is served, which results in a very fast response. However, the problem with the fall back occurs when the network signal is very weak. In this situation, the device will try to download the file because it can detect that network connection exists, but what it fails to do is see that the signal is weak. This slow response time leads to a very bad user experience. What is interesting in this situation is that the alternate version of this file we want to download is already on the device but the application still tries to get the file from the server.

Any modification to the manifest file will result in the application being refreshed. This might not be desirable because files that will that might not have changed are downloaded again. Another problem with the App cache is that if there is any error with one of the resources, then all files will fail. Developers have to be careful when modifying the content of the manifest file.

3.6.2 Improving App Cache

A much as the App Cache presents an easy way to enable web application to work offline, there are too many assumptions that it makes, and as a result might not be that efficient in some situations.

So instead of having to enable a web application to work offline in a declarative way like the current App Cache does, it will be good if the API could provide a means for the developer to use a client scripting language like JavaScript to explicitly code the flow of events. For example, when the window loads we can register a script that we want to handle the offline mode scenarios.

```
<script>
  window.register(``*'', ``controller.js''>
</script>
```

Or simply use a declarative way to specify a script file we want to run register.

```
<html pageController=``controller.js''>
```

In *controller.js*, we can explicitly specify what we want to cache. First we list the files to be cached. We then hook to an *onUpdate* event in order to cache the files when there is a change. When a request is made to load a resource, an *onRequest* event will fire that will be used to fetch the requested resource from the catch, and if nothing is found then the page request process can proceed as normal, which is by getting it from the server.

```
cacheFiles = [
  ``favicon.ico'',
  ``index.html'',
  ``stylesheet.css'',
  ``images/logo.png'',
  ``scripts/main.js'',
];

appCache.onUpdate = function(event){
  appCache.cache(cacheFiles);
};
```

```
appCache.onRequest = function(request){
  var resource = appCache.get();

  if(resource){
    request.setResponse(resource);
    request.preventDefault();
  }
};
```

This gives us the ability to specify exactly what we want and how we want the resource caching to be handled.

3.6.3 Leveraging Browser Cache

When pages are downloaded to the client machine, browsers can automatically store these pages using the browser caching mechanism. However, the browser can try to re-download these files even though they might not have changed. To make sure that the pages are stored for a longer period, an expiry data can be set on the HTTP headers to let the browser know that it does not need to fetch the resource again until the expiry date is reached (62).

Some of the major browsers like Google chrome and Internet Explorer use a heuristic in order to make a decision on how long they can store the files for, if an expiry date is not set on the HTTP headers. Both HTTP and HTTPs pages can be cached although older versions of browsers will only cache HTTP content.

In order to make sure that we take advantage of caching on all web browsers, the web server must be setup to set expiry dates on all resources that can be cached such as CSS, JavaScript, HTML and images files or other binary resources like video and audio.

HTTP/1.1 Caching Response Headers

`Expires` and `Cache-Control: max-age` stipulate the duration in which a web browser can consider the content up-to-date before asking the web server for a newer version of the resource.

`Last-Modified` and `ETag`. In this case the browser will issue a GET command to

the server and the server will check the resource to see if it has been changed or is still the same as that stored at the client. With the `Etag` header, a unique identifier or version number is set on a resource. The resources are only downloaded to the client if this unique identifier has been modified.

To make sure that you limit any unnecessary downloads, all static files should have their cache headers set. The duration of the cache should be dependent on how the application gets used and how the contents frequently changes. However having a minimum of one to twelve months is a good start. As much as the expiry date of a resource can be set to more than twelve months, it is nevertheless not recommended. It is highly recommended to use the Expires headers as most of the browsers support it compared to the `Cache-Control: max-age`. Most of the web browsers delete their cache based on a process that checks to see when the resource was last used.

Fingerprinting Frequently changing resources

There are some files that we want to cache even though they change frequently. These can still be set to have their expiry date way in the future, but can have a "finger print" that will enable the file to be re-downloaded when a more recent version is available. This can be achieved by this by adding a "fingerprint" or unique identifier to the URL of the resource. Whenever the server wants the client to download the files again, all we do is to change the fingerprint. When the finger print changes this also changes the URL, hence the web browser recognizes this as a new file and will force the re-downloading of the file.

Leveraging Proxy Caching

We have been taking about net unavailability in relation to the client not being able to access any network. There are, however, some cases where the user might only have access to access to certain networks. If the application the user is trying to access is on an unreachable server, this is normally a good candidate for proxy caching. Resources can be cached or made to be downloaded from the neighboring proxy server instead of getting it from the remote origin web server (68).

Proxy caching is good in that even web users who have never downloaded the web application from the origin server can still gain access to the application through the proxy servers. Proxy servers are usually closer to the web user and therefore, this can lead to a substantial decrease in network latency thereby increase in performance.

Cache-control: a public caching header can be used specify that a file is available for caching by both the requesting client browser and the public web proxies.

Proxy Caching Considerations

- It is important to note that some proxy servers do not cache resources that have a query string as part of the URL. This limitation can be overcome by encoding the query string into the requested page filename.

- When using proxy caching, some proxy servers do not cache pages that use cookies. For those that do, all cookies that are set by the pages are accessible to all the web users, as a result the cookies should not be set for individual use.

- Some proxies cannot handle `Content-Encoding` properly and this can lead to files being delivered to the client in a format that will not be usable e.g. sending a compressed file to a browser that cannot decompress the file. The use of Cache-Control header for such a resource should be set to private so as to stop the proxy from caching the resource.

- The proxy can store different versions of the same file when `Accept-Encoding` response header is set to *Vary*, for example, if a proxy can store both compressed and uncompressed versions of the resource then in the end the clients that can handle the compressed version will receive compressed version while those that cannot will receive the uncompressed version (69).

3.7 Designing a Fluent Interface

The term Fluent Interface was first introduced by Eric Evans and Martin Fowler as a programming technique that aims to present an API in the most simple possible way while making the API easy to use by improving readability. Method Chaining is an important technique used when designing fluent interfaces. API's are normally designed in such a way as to allow a generic way of programming and as it evolves the ways it gets used may become obscure. Fluent Interfaces can allow the API to be used in a clean and simple way, offering a more clear descriptive way that improves its usability (70). In some cases, in order to perform a simple task using the IndexedDB API, one might need to write a sizable chuck of code. Let's consider the following code to use a certain criteria, select a set of *Student* records from record from the *StudentsDB* database.

```
var request = window.indexedDB.open("StudentsDB");
    request.onsuccess = function(event){
        var db = request.result;
        var transaction = db.transaction(["Student"], IDBTransaction.
            READ_WRITE);
        var objectStore = transaction.objectStore("Student");
        var request = srv.objectStore.openCursor();
        request.onsuccess = function(event){
            var result = request.result;
            if (result) {
              write(result.key + ": " + result.value);
              result["continue"]();
            }
        };
    };
```

Listing 3.4: Traditional Way of Querying IndexedDB

The code above is doing a very simple task but requires us to write a couple lines to get what we want. It will be much easier and productive if we could simplify the the interface to something like:

Using a Student in StudentsDB, Select and loop through, print details.

Using Javascript, this could simplified to;

```
FluentIndexedDB.from("Student")
               .in("StudentsDB")
               .select().
               .forEach(write(result.key + ":" + result.value));
```

Listing 3.5: Proposed Way of Querying IndexedDB

The creation of this fluent interface is archived by implementing method chaining. To facilitate the creating of a fluent API, an interface can to be defined. The calls to the public method must return the reference to the object itself. This helps in chaining of the object function or operations.

3.7.1 Domain Specific Language (DSL)

"Domain-specific language is a computer programming language of limited expressiveness focused on a particular domain." (76)

The concept behind a domain specific language is to have a programming language that is directed at a specific problem, instead of being a general purpose language that can be used for solving any type of problem. Of special interest to this thesis are internal DSLs, which refer to customizations made to a language in order to make language feel like another language and is often referred to as a fluent interface. DSLs can either be text based or graphical.

To get more out of the meaning of Fowler's definition we are going to look at the following key elements:

- **Computer programming language**

  We use a Domain Specific Language to give instructions to the computer in a structure that is easier for us to understand but at the same time executing what ever it is that we want the computer to perform.

- **Language nature**

  A DSL is a language, and hence must have some form of fluency in the way it

is expressed and the meaning of the expressions should come from the way the expressions are composed together.

- **Limited expressiveness**
  When we look at programming languages that are not limited to a domain, they usually allow a number of of capabilities which normally will make it harder to grasp. A DSL should support a very small number of features only enough to support a given aspect of a system or domain. DSLs are not meant to be used for building a complete applications (76).

- **Domain focus**
  Concentrating on a narrow aspect of the domain will make the language more useful.

The need for a domain-specific language (DSL) is to plainly express statements in a particular area or domain. CSS and SQL are good examples of domain specific languages. The IndexedDB API is intended to be used for data manipulation on the client machine. This is a specific area and specific use, but IndexedDB is designed in such a way that it leaves to the developer to implement their solution using JavaScript. This can be powerful but unproductive as a lot code is required to perform a simple task. In general, a domain specific language can be used to automatically generate code.

jQuery is an established JavaScript library that can be perceived as an internal DSL for selection and manipulation of DOM elements. jQuery supports the creation of extensions and plug-ins and we can employ this feature to add support for querying IndexedDB.

Advantage of creating domain-specific languages:

- It comprises of constructs that are precisely suitable for a problem domain.

- A domain-specific language is made up of elements and relationships that directly represent the logic of the problem space which makes it easier to discover and fix the problem with the logic.

- It can easily be used by domain knowledge specialist who might not be developers. When the DSL has a graphical representation of the domain, it makes it easy for people to visualize and understand how the application is modeled.

3.7.2 Method Chaining

Method chaining is an object oriented programming language technique in which the methods in an object always return an object which can possibly be the current object itself, allowing methods to be connected in a single programming expression (77). Method chaining is sometimes referred to as "train wreck" because of the large operations stacked together (94). When designed well this helps to reduce the amount of code and greatly increase program readability and now widely associated with designing of Domain Specific Language. Method chaining greatly helps in picking the clauses that fit a particular situation, making it a great way of building expressions (76).

JavaScript uses a semi-colon as statement terminator and this is important as it gives us greater flexibility when writing expressions because it allows us to have expressions that can span multiple lines which can help in increasing readability. This is an example of how Method Chaining can be used.

```
return (new int[] { 5, 4, 3, 2, 1 })
    .AddRange(new int[] { 6, 7, 8, 9, 10 })
    .Sort()
    .Reverse()
    .ForEach(i => i + 1)
    .Insert(0, 0);
```

Listing 3.6: Method Chaining Example

3.7.3 Language Integrated Query (LINQ)

Microsoft developed LINQ in order to add querying capability into mainstream general purpose, statically-typed languages language like C# and Visual Basic. When working with data, it is easier to use a query language to query or manipulate data. With a general purpose language, we might need to write loops to iterate through the data to find or manipulate the required information. With LINQ, developers have an easier way writing queries against object collections. LINQ provides query or sequence operators which work on query expressions using translations rules for data projections and filtering against object collections.

LINQ describes a set of function names referred to as the standard query operators, or standard sequence operators, in conjunction with interpretation rules from query expressions to expressions using these function names, lambda expressions and anonymous types.

To get a list of students whose age is less than eighteen using C# 2.0 the query would look like this

```
var query =  from s in Students
             where s.Age <= 18
             select new {c.Firstname, c.Lastname};
```

To present this expressively using a fluent interface, it will look like this;

```
var query =  Students
                .where(s=>s.Age <= 18)
                .select(s=> new {c.Firstname, c.Lastname};
```

The advantage of using the LINQ syntax above is that it is very similar to the way JavaScript code is written and as a result will be easier to implement.

Lambda Expression

A lambda expression is an anonymous function (71), you can put inside it expressions and statements and you can execute that lambda expression or you can pass it around as a parameter. So, when we are writing LINQ queries with the operators such as *where, select or orderby* we can pass in those lambda expressions

to those operators, for example, we can pass in a lambda expression to the *where* to specify how we would like to filter something.

3.7.4 Lazy Load Pattern

The lazy load pattern is generally used to increase the performance of an application by reducing the amount of work it needs to get data when that data is unnecessary. When you have an application where you work with objects that need to be persisted from an object store such as a database and you want to be able to load portions of the object that you don't necessarily need at this point in time. The reason why we want to avoid it generally is to increase the performance of the application. This is especially important when you are talking about portions data that require a separate request to fetch and or which are just simply expensive to retrieve. Ideally we want the client code to be ignorant of the fact that the object is performing this lazy loading. The lazy loading pattern defines several strategies to achieve this goal.

"The lazy loading's intension is to interrupt the object loading process, for the moment leaving a marker on the object structure so that if the data is needed it can be loaded only when it is needed." (75)

3.8 Data persistence

When working with offline web applications system, there are two approaches to data persistance that can be considered.

1. **Pessimistic Approach**
   In pessimistic approach, the web application assumes that if updates are made then there is most likely going to be conflicts so in this approach, data modification is not permitted. Data is only used for reading.

2. **Optimistic Approach**
   With an optimistic approach to offline data usage, the web application will

allow data modification, hoping once the application comes back online, the data can be synchronized with the web server with no data clashes.

In order for application to work fully while offline; this optimistic method is the best approach. However, if there are any conflicts caused by the modification of data, the application will need to detect and handle this appropriately. In this way, the data created or used by one user can be shared with a number of users at the same time.

3.8.1 Integrity and Reliability

Data updates must be performed in a reliable way even in situations where instant data consistency is not a strict prerequisite. For instance, if the web application implements an online shop and a customer orders a product through the system while offline, then the ordering process must still be completed and the order honored. The application must be designed in such a way that when the system gets back online then a synchronization process is automatically run in order to reconcile the differences between the web server and the client. In cases of the synchronization procedure failing, then data should not be lost and the process should be able to be resumed.

Therefore, any solution that replicates data between databases must implement a reliable mechanism for transporting and processing this data. If some cases where the system takes care of this synchronization process is unsuccessful, it must be able to restart this procedure without losing or replicating any data.

3.9 Database Synchronisation

When using offline enabled web application, users have fast access to their information and have no need of network connection every time access to data is required. During offline mode all updates are done locally and will need to be synchronized with the central storage once the network is restored. As much as storing data locally resolves the problem with network unavailability, there are some issues

associated with this model. In this section we discuss some of the challenges and suggest techniques that can be used to evade these problems.

3.9.1 Data Consistency and Application Responsiveness

Having data consistency and a responsive web interface are two contradictory requirements that developers need to pay attention to. In order to have high degree of data consistency between the server and the local database, then the application must be designed in a way that stops different clients from requesting or modifying data.

If you call for a high degree of consistency across all databases, then you must take steps to prevent competing applications from accessing data that might be in a process of change. However, this is difficult with web applications because since HTML is a stateless protocol, once a user requests information, they get disconnected from the web server so we cannot impose locks on retrieved data.

When a snapshot of the server data is cached or stored in the local database, this will help increase application responsiveness but at the same time compromise data integrity and consistence.

3.9.2 Client-side Transaction Logging

When an object is changed, a transaction log can be created at the client side detailing the changes that occurred. Once the network becomes available, the transaction log is then sent to the server or for automatic or manual reconciliation.

3.9.3 Receiving Server Updates

Clients can receive data from the server in two ways. The first time an application is run in the client browser, the server sends all the data that is required by the client. In this case there are no conflicts to worry about since all this is new data.

The second way is when the server sends information to that client that already has some information on their device. In this case there is need to know how old or

fresh the data at the client is before an update can even be made. The database or object stores at the client-side will need some sort of versioning in order to see if an update is required and also to make sure we only download what changed rather than downloading everything.

In order to download data from the server and leave the web application responsive, Web sockets and web workers can be used. Web sockets support full duplex communication. A channel can be can be created and be left open, to allow the server to broadcast messages or data updates to all subscribers or web clients.

3.9.4 Change Tracking

In order for locally modified data to always reflect what is at the server a way of tracking the changes is necessary. With change tracking all data modifications including inserts, updates, and deletions are record. If we take an example of user wanting to request the latest data stored at the server, in order for them to have the most up to data. If the data modification is not tracked, it means the user might end up overwriting changes made by other users. It might also require the user to download all the data stored at a remote server. This might be very inefficient since a huge amount of time may be needed to retrieve information if there is a high volume of data.

- In order to avoid these problems, keeping track of the data modifications is very important. Changes can be tracked by using row or record versions. In this case every table in the database has a column that represents the version number of the row.
- When the row is inserted into the database the version number is set to a number that represents and new row for example, version number could be set to 0. The row version number gets incremented by 1 every time it gets updated. Below is a scenario of a read write operation.
- Read a single row from the databases, for example, with version number 1

- The client makes changes to the row
- The client sends a request to update the modified row in the database.
- The server service uses the key of the submitted row and performs a read on the same row before committing the changes.
- If the row version number of the modified row is the equal to the row version of the row retrieved by the server service then the version number gets incremented by 1 and the change is committed. Otherwise if the version number of the row is not equal to the server row version then it means the row was modified by another user or process and the process is rolled back.
- A response is then sent to the client with the result of the operation.

3.9.5 Conflict Detection and Resolution

Change conflicts are one more issue that is associated with offline web applications. This happens when two or more users or services modify the same information and then attempt to apply the changes to the database at the same time. For instance, when two users modify the same customer's contact details to two different values, the first user then successfully synchronizes with the central repository, a conflict will arise since the existing version of the record is not the same as what was expected by the synchronization service.

There are a few of techniques to solve these clashes. For instance, the most recent modification to be sent to the server can be assumed to be the one that wins, or on the other hand, it possibly can be based on which of web application users has the higher role.

Another form of conflict occurs when a part of the row to be updated is stale. For example, a table consisting of 27 countries in the European Union (EU) is stored in local storage at the client browser. Let's assume Belgium, which is one of the countries in the EU decides to leave the union, and get deleted from the member list at the server side. If an offline web application user decides to modify or insert

a row with Belgium in the country column, this will be rejected at the server side as the country will not be a valid on hence the local storage data is now stale.

When designing database tables, it is important to carefully consider how table columns are defined as it can affect the way data conflict are resolved. For instance, if we create an IDENTITY column, we should make sure that the algorithm for generating this identity values will not result in a conflict during synchronization. Usually identity columns are generating by simply incrementing the seed by one. When different users insert the rows during offline mode, the values generate could be the same which will result in a conflict. So instead of using IDENTITY columns, we can use GUIDs, as they are certain to be unique.

When the data synchronization process runs, the clients send all changes made since the last data retrieval to the web server. This process should also include downloading new data from the server and updating the local database.

Any data conflicts can be resolved using the following conflict resolution strategies:

- **Web Server Wins**

  If the data at the server has changed since the last update the server data is automatically downloaded to the client and overwrite the data at the client.

- **Client Wins**

  In this scenario any changes to the data at the client are applied to the web server database. This differs from the web server wins strategy, in that most recent data synchronization to the server always prevails.

- **Highest Version Wins**

  In this is case every record maintains a version number, and during data synchronization the version are compared and whichever has the higher version predominates.

3.9.6 Prioritizing Data Exchange

When deploying offline enabled web application it is important to examine ways in which the exchange of data can be boosted as the transmission channels might not be available or might have inadequate bandwidth. Identifying and then classifying data based on its priority is a good approach to enhance data synchronization. With this method of data synchronization, information that of high priority or any critical modifications can be synchronized with the server immediately or as soon as connection becomes available, while data that is of low significance can be done at a later time.

3.9.7 Background Synchronization

It can be very upsetting to a user to have their user interface blocked when a different process (for example, data synchronization service) starts running. This synchronization service should be designed to work in the background. The user should be able to continue with their work even when the synchronization service is running.

3.9.8 Security

When developing offline web application, there are some aspects of security that need to be considered;

- The database might need to be encrypted.
- To be able to have access to the database, some form of authentication is required.
- The synchronization service should be able to work with encrypted data or should be able to scramble the data before sending it to the remote database.

3.9.9 Error Handling and Recovery

The data synchronization procedures must be designed in such a way that any errors that occur or bad data are handled in such a way that synchronization of any good data that occur will still continue.

If the data synchronization process is designed as a background services, then all errors that are encountered during data synchronization process must be logged on the remote server instead of being presented to the client. These errors can be logged by using error log files, system even logs, or by creating a database table to record the application errors. Depending on the errors encountered, the process can also be designed in such a way that it can systematically fix the errors, and repeat the synchronization process instantly or after a specified duration. If the system is not designed to run as a background process, then the errors can be presented to the user for them to decide on what to do or for them to manually fix the errors before re-submitting the data again for synchronization.

Chapter 4

Privacy and Security

4.1 Privacy

4.1.1 Introduction

With the advancement in internet technology, most of our day to day services are now available online (98) and as a result, the web application complexity is increasing. However, most of these web applications require the customer to enter their personal information into the system in order to gain access. This information is usually sensitive information. For example, user login details might include personal emails, bank account details or medical details. This development has had an enormous impact on the security and privacy of user information.

User privacy needs to be carefully considered and complex security measures are required as user information is now available in a distributed architecture.

4.1.2 Privacy By Design

The principles of "Privacy by Design" is a recommendation from the W3C that requires developers to take the initiative of considering privacy as an important part of designing web application (99) without making compromises or without waiting to fix problems when they happen. The User privacy must be treated as the default setting in web application design.

4.1.3 User Centred Design

Web applications are used by people, so it is imperative that the design should be centred on the user (100), providing them with the confidence that they have full control of their personal information and know in what way it will be used.

The use of user information needs to be transparent and users should be able to carefully decide whether they want their information to be shared with other third-party applications. Furthermore, users need to know the purpose of their information and the duration their data will be used by the application.

In some case an application can keep personal information as a way of improving user experience, for example, an application can store a user's payment information to make it easy for the user to make future payments. In this case, the application provider has to specify the information that will be stored and the associated risks. The users should be allowed to review, modify or delete their information.

The application must only collect relevant data, and should never collect more than is needed to provide a particular service.

Sensitive user information should be encrypted wherever possible and should any ill use of data occur, there should always be corrective measures in place.

4.1.4 User information confidentiality

User information confidentiality can be preserved by using secure channels like HTTPS when transmitting data over the internet instead of using HTTP. The information also needs to be stored in a secure database in order to safeguard against unintentional or deliberate loss which could be through malicious attacks.

4.1.5 Control and log access

Access to all personal user information must be tightly controlled and any access to the data must be recorded. This is an important measure, to make it possible to track what happened in case of data loss or misuse.

4.1.6 Sensitivity of data

All web users' personal data is considered special and must be handled with care, particularly when its misuse may perhaps lead to privacy related issues like loss of income, identity theft or unlawful distribution of someone's personal information. When designing web application, we need to classify data in terms of the impact on the disclosure of a piece of information and identify how best to secure it.

4.2 Security

4.2.1 Introduction

Web applications are now a major supporting part of businesses and organizations in every part of the economy. They help associate workers, consumers and businesses to the information they need in their day to day lives. This has led to the reduction of costs and has radically increased the speed of doing business. However, this has also increased the risk to users' data and malicious attacks are being used every day, directed at web applications in order to affect business processes.

When designing web applications, a careful consideration of security threats, by identifying weaknesses, threat entry points (72), and all potential users of the system is an important step. The analysis of the security problems, designing of mitigation approaches, and evaluating solutions to these problems can be accomplished by isolating the attackers, assets, threats and other components. These threats can be categorized so that, those that pose the greatest threat are carefully catered for by choosing mitigation approaches and by building solutions based on strategies.

Developers basically need to pay particular attention to everything that the users enter into the system. Any web application that accepts user input is susceptible to attack (85). Two of the most usual attacks against web application are SQL injection and cross site scripting. We should always make sure that user input is well formed. The system should be designed in such a way that it should be able

to validate input as it is entered into the system. For instance, if we are expecting an integer to be entered, we should parse the input into an integer. If the parsing fails, then we should assume that the input was malformed. If the input is a string value, we can use regular expression *(Regex)* partten matching to see if the input matches the expected values.

A scalable solution should be able to provide a secure web application without compromising application performance (86). It should be accessible to its intended web users and at the same time avoid accidental data loss, identity theft, phishing, SQL injection, application Denial of Service (DoS) and other malevolent attacks.

In case of an attack, a web application should always continue to perform well and at the same time it should continue to be accessible to its users despite the level or extent of the attack.

4.2.2 Injection Attacks

An injection attack is considered as the most common security attack in web applications (87). Some of the examples of injection attacks are Cross-Site Scripting (XSS), SQL Injection, Header Injection, Log Injection and Full Path Disclosure. These forms of attacks can be very hard to safeguard against.

Injection attacks take place when malicious data is included in a command or query and used to run queries or commands that are not intended, in order to modify or gain access to unauthorized information.

SQL injection

This is by far considered to be a very common and the most dangerous form of inject attack (88). SQL injection attacks exploit input validation weaknesses that might be present in a web application page by sending unauthorized SQL queries to the remote database. Malicious attackers use SQL injection to infiltrate the remote database and steal information. With SQL Injection, attackers inject mali-

cious information into a web application page form or through the page URL which then gets used in SQL queries in a way that was not originally planned by the developer. In some cases, data can come from the application database and developers would typically trust data from their own database. However, data that is safe for use on one page might not be safe for use on other pages. All the data that comes from the database and needs to be presented to the user, should always checked and appropriately encoded.
Let's take a look at the following PHP code as an example;

```
$db = new mysqli('localhost', 'username', 'password', 'database');
$result = $db->query(
    'SELECT * FROM transactions WHERE user_id = ' . $_POST['user_id']
);
```

There are a few issues with the script presented. To start with, *$_POST('user_id')* is not being validated to make sure it is a valid entry. In addition, we are using data that is not validated to give us a user id to use, of which an attacker might use any value the want, and as a result, they can get transaction information about anyone, as long as they provide the correct user id. On the other hand, the *user_id* is not escaped to make sure that malicious content has not been added to alter the way the SQL statement was originally meant to execute as the input was never validated.

Attack Using Data from the database

Data retrieved from the database should also not be trusted. For example, when updating a table of users, it is common that some first or last names have single quotes. If these quotes are not escaped when the records are being created, an attacker can store a SQL string in these database columns which will result in the value being altered to execute unintended scripts as the data is being read from the database.

It is important to note that IndexedDB is an object oriented database and does not use SQL to read or write data; as a result it is not prone to these kinds of attacks.

Modifying SQL statements might be intended but not limited to the following;

- Leakage of data
- Release of unauthorized information
- Modification of information stored at the server
- Avoiding authorization controls
- Client-side SQL Injection

Protection from SQL Injection

To safeguard against SQL injection, data should always be validated to make sure that is in the expected form and should be escaped before it is included in a SQL statement. Most SQL databases allow the use of parameterized or prepared queries and are a safer way to avoid SQL injection attacks, so if possible escaping of SQL queries should always be avoided as escaping a query can never be enough.

Implement Least Privilege Principle

It is important to stop SQL injection from happening than to allow it to happen then work on stopping the injected queries from executing. When hackers manage to run SQL queries, they do this as database user. When implementing the least privilege principle, only the minimum rights a database user requires to successfully perform their work are provided (90). The web application should never gain access to the database as the root user or a super user. One different way of implementing this principle is to have distinct roles, one to perform database reads and the other for performing database writes. This will make sure that when SQL injection is used, a user that can only read the data then they will not be able to perform a write and the other way around.

Input Validation

Input Validation is the application boundary that acts as the barrier that shelters all the other layers in web application. Anything that the user inputs into the

system should be taken as a possible threat to the application and should be validated. Most of the problems encountered by a web application are a result of input validation failures, hence it is important to design the system well.

Cross-site scripting (XSS)

Cross-site scripting attacks exploit script injection flaws in a web application page and redirect the details posted by the user from the page to the attacker.

Cross-site scripting (XSS) is a form of web application security vulnerability through internet browser security breach that allows malicious web sites to inject client-side scripts into a page from another website (87). According to a Symantec report, 84% of the online security vulnerabilities were as a result of cross-site script attacks (97). There is need to implement very high security mitigation techniques if the information in the web application is highly sensitive or confidential.

Non-persistent

The non-persistent is a very common type of cross-site attack (91) which normally occurs when data is requested using parameters in the URL. If the server does not properly clean up the request to make the format of the data that is as expected, this can become a problem if data in the request needs to be redisplayed in the browser. If the data in the URL contain malicious scripts and the HTML mark-up was not appropriately encoded to remove control characters, this will lead to mark-up injection. Redisplaying of a string in the request is normally common in situations where the request is a search operations as it will be important to display what the user has provided as part of the search operation.

Persistent

This form of cross-site scripting attack is a more permanent variation. With persistent XSS vulnerability, when a script is injected into the request, the server stores the malicious script without properly encoding it, (91) meaning the script will get executed every time the page is displayed. Discussion forums are a good example, as users are normally allowed to enter comments which are then get saved and displayed every time the page is requested by other site users.

Same-Origin Policy

An origin refers to the scheme, host, and port of a Universal Resource Locater (URL). In web applications development, the same origin policy is intended to act as a data access security measure for client-side languages like JavaScript. The policy allows scripts running on a specific page to be able to call methods or act on properties provided by another page on the same site. Accessing methods and properties across different websites is not allowed (92).

Web pages coming from different sites are isolated. For example, the script on the page `http://example.com/page1.html` will not be allowed access to the script on the page `https://example.com/page2.html`, because the origin of the first page, (http, example.com, 80), is not the same as that of the second page (https, example.com, 443).

This separation of data is very important in offline web applications in order to maintain data integrity or prevent data loss either accidental of due to malicious attacks.

The same-origin policy differentiates between the information being sent or received. Generally, one source is allowed to transmit data to another source, but is not allowed to receive data from another source. This prevention of receiving information is put in place as a way of stopping malicious web sites from working on information that might be confidential and not intended for them. Under this policy, sending of information across different can be unsafe since it allows attacks for example, cross-site request forgery (CSRF) and clickjacking.

Cross-Site Request Forgery Cross-Site Request Forgery (CSRF) is a way of misleading the user into opening a web page that has a malicious request. It is bad because it impersonates the user and performs malicious operations such as changing the user's personal details (73). CSRF attacks are normally used for changing data at the remote server or for accessing sensitive information.

Web Storage Vulnerabilities

A major security issue with web storage is that web users do not know what the information gets saved on their local device. There is no way for users to choose what they want to store or read from the local storage. All the writing and reading of information is handled by the web application client scripts. The positive thing about web storage is that it implements the same origin policy and as a result scripts from other domain cannot access the data. However, malicious code can be run through cross-site scripts if the application is not properly designed to safeguard against such attacks.

Session hijacking

Malicious attacks can occur through the use of session identifiers. Let's take an example where the session identifier is stored on the user's device and an attacker manages to inject some malicious script into the application in order to steal the session. If the application uses the session identifiers to identify the web user and their session then gaining access to this value (74) will mean the application can serve information to the wrong user.

If an attacker manages to inject the following script, a session identifier can easily be retrieved.

```
<script>
document.write("<img src='http://www.example.com?cookies="+document.cookie+"
    '>");
</script>
```

However, HTML5 web storage implements this in the same way, only this time the attacker has to know the name of the variable used to store the session identifier which adds a level of security.

```
<script>
document.write("<img src='http://www.example.com? sessionID="+localStorage.
    getItem('SessionID')+"'>");
</script>
```

In order to safeguard against stealing of cookies, cookies can be set to use HTTP only so that attackers cannot be gain access to them using JavaScript. However, HTML5 local storage poses a problem here because it does not make use of the HTTP Only flag.

4.2.3 File System API Security Considerations

When working with file system API, there are a number of security and privacy issues that need to be considered. The API might permit a script that is not trusted to read or write files to the local file system:

- Denial of service attack
- Untrusted scripts can write too much to the file system and fill up the hard disk or can use too much bandwidth. This can partly be moderated by imposing quota limitations.
- Private information can be deleted or stolen. This can be corrected by adding a limit to the way the script read or write to the local file system, only scripts from the same origin can share files.
- Scripts can store malicious files or content that is illegal, on the local file system. This is the same as any general download, and similar safety measures still apply, however, this is possibly worse in that:
  - It possibly will consist of several files.
  - The files may possibly be in a portion of the file system that is more difficult for the user to find than what is normally the case with normal downloads directory.
  - A untrusted script might gain access to write to the file system long after the user has been allowed access to a trusted script as they might not be able to the distinguish the two separate events.

These risks may be reduced by limiting the creation or renaming of files that do not have executable extensions. Developers should try to ensure that filenames are harder to guess and the API should not be used to set the extension of the file.

4.2.4 IndexedDB Security

IndexedDB makes use of the same-origin policy, which means it restricts the database access to a domain or site that created the database and scripts from other websites will not be able to read or write to the database. It is also not possible to read or write to the IndexedDB database from either the *<frame>* or the *<iframe>*.

4.2.5 Web Sockets API

Processes can be made to run in the background whereby an application initiates a web socket connection between different domains. All this can be done without requesting authorization from the web user who might not even release it. Malicious script can be made to run in the background to still or modify user data and at the same time consume user's bandwidth.

4.2.6 Countermeasures

Although the HTML web storage comes with some advantages, it also has some problems associated with it. When designing offline enabled applications, security of user data needs to be carefully considered.

The use of Local Storage brings many advantages but also gives way to some of the common attacks discussed above. There are several things that may possibly not go according to plan, so we need to carefully design access to local storage attributes. In order to use local storage without compromising security we need to think through the following;

- Use session storage in place of local storage when managing session data

because the session data served using session storage will only be available for the duration of the session.

- Using HTTP only flag for cookies can also prove to be a better option as JavaScript will not be able to access these cookies.
- We should try by all means to refrain from storing sensitive information in local storage. It is safer to have sensitive data on a remote web server where we have more control.

Chapter 5

Discussions And Conclusions

5.1 Discussion

While the web application design techniques presented are plausible, there still are many challenges in the development of offline web application.

There is no easy way to control the data that is entered while the user is offline as users are able to delete temporary internet files on their machines at any time without any way of warning them that data could potentially be used in one of the offline applications. It will be nice if there could be a way for users to identify a database they do not want to delete before clearing their browser cache. In some case the users also do not have any clue that they are in fact running an offline application unless application notifies them of this.

There is no control over the location of data due to the different implementations of different browsers and platforms by different vendors. All browsers store cache files in different locations, which means if a user opens an offline web application and saves data in local storage, for example, using Internet Explorer (IE), if they decide to open the same application using Google Chrome, they will not gain access the information they would have saved in IE.

In order to allow synchronization to work the application has to be opened in the browser as there is no automatic or behind-the-scene synchronization, for example, as with windows services. This is because, all JavaScript execution only

happen when the browser is opened.

Storing data at the client-side has the risk of running out of disk space.

Conflict resolution and handling of errors that are encountered during data synchronization does however present a good foundation for future work.

5.1.1 Design Considerations

- **Study Web Application User Actions**

  This is one of the most important considerations. Having the knowledge of how users use the application and what actions the user take and the frequency at which the actions occur will help in figuring out the pages that require optimization and the ones that can be used offline. This can be archived by looking at the server log or where possible identifying a group that can used as volunteers.

- **Make sure every page loads faster**

  It is important to make sure that the web application loads faster. Making a web application available offline does not mean users have to suffer the slow paging loading when online while the application is busy preparing itself for offline use. We should always make sure that any background loading of content or pre-fetching of resource is only done when the page has fully loaded and should have a very limited impact on application performance.

- **Storing the right data**

  Not all web page content is the same. There are other resources or content that are not safe to store on the client browser, especially if the application can be accessed from the public domain. Most static resources can be good candidates as they take longer to load and are safe to store on the client computer. Some information is time sensitive and requires frequent updates in order to have the correct information.

- **Be a good web citizen**

  Most users usually run a number of applications concurrently, so we should

make sure that the web application does not use too much of the user's bandwidth. Only store the information that you are sure is most likely to be required during offline use.

5.1.2 Implementation risks

Obstacles to adoption While offline web services indeed enable the client to work in a disconnected environment, there are several issues that developers will need to consider.

- **Security and privacy**
  If the information stored in the client browser is private or sensitive, is this information safe from malicious usage?

- **Regulatory compliance**
  Does serving data at the client computer which could be accessed publicly, meet regulatory requirements? There are some industries like banking, health services and governments that put stringent rules to service providers to make sure that they meet the necessary requirements.

5.1.3 Browser Differences

The W3C is a body that leads the development of open standards to ensure the long term growth of the Web. It is the body that tries to make sure that browser vendors adhere to some standards thereby enforcing compatibility. It is however difficult as often some standards take time to be finalized and this results in vendors producing versions of HTML that are not compatible. This is a problem area for developers as they have to find ways to work around these incompatibilities so the application can work across the different browser implementations. This often reduces developer efficiency due to the increase in the development time which also leads to increased costs.

5.1.4 Performance and Scalability

What developers need to realize when developing in web applications is that as much as the application looks like a basic desktop application, it is essentially a distributed application (84) that might be used concurrently by several other users. Depending on the web application processes running, high performance might be required and this in turn calls for high scalability.

A web page must be served to a user in the same way as it should be served to several other uses concurrently. The application performance must double when the web application is running on a computer which doubles the resources. Nevertheless, this is what is expected in principle but the closer the application is to this the better the application will perform when subjected to a bigger workload.

There are a number of ways to measuring application scalability. For a web application to be deemed scalable, the following should be considered:

5.1.5 Security

Adding security measures usually affects application performance and scalability. For instance, if we implements a page with a form for users to enter details, adding input validation will mean the page's performance will be slightly be reduced compared to sending the input to the server without checking it first. Nevertheless, application security procedures must always come first, irrespective of how unlikely we might think the web application is prone to malicious attack.

5.1.6 Client-side Development and Debugging

Most major browsers come with client side developer tools which include JavaScript debugging tools preloaded or can be installed as browser extensions. These can be used for examining the contents of web storage, indexedDB database, other than the HTML, CSS, and JavaScript that puts them all together.

5.2 Summary And Conclusions

In order to create responsive web applications, it is important to look at both, the server and client side components. A well built code base, web server and client caching techniques, web page load acceleration through resource compression aide in the delivery of an optimal, high performance and highly responsive offline enabled web applications that lead to a better user experience.

Normally, web applications require a network to function, however, network availability is not always guaranteed and web users always want their applications to work regardless. Designing an application that anticipates this lack of network connectivity will help ensure that it will always be available and will work better in most web user circumstances.

In this thesis, we looked a number of ways we can design rich web applications, that will work both online and offline, an effort that is geared towards providing a reliable user experience. We presented the HTML5, IndexedDB, web storage, Web Workers and Web Sockets APIs. It was also noteworthy to look at IndexedDB, a new object oriented database that will greatly help in bringing a database to the client browser thereby greatly improving application performance and user experience. We also looked at data synchronization techniques to ensure that, the data that is created or modified and stored at the client-side database is reconciled with the data at the server.

HTML5 is still in draft mode, although most of the HTML5 specifications have been implemented in the majority of the major browsers. However, in some cases, some of the specifications are not implemented in the same way.

HTML5 is a game changer. It brings a lot of features that bring the web application closer and in other cases above their desktop counterparts.

As we bring user data and business logic to the browser, the privacy and security of user data are some of the issues that require careful consideration.

Bibliography

(1) John Kanalakis. *Developing .NET Enterprise Applications*. Apress, August 21, 2003.

(2) Livermore, L. *ParFlow Project Overview*.
Retrieved from https://computation.llnl.gov/casc/parflow/overview.shtml. John Wiley And Sons, March 15, 2011.

(3) Lubbers, P., Albers, B., Salim,F. *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*. Apress, Sep 1, 2010.

(4) Apers, C., Paterson, D. *Beginning iPhone and iPad Web Apps: Scripting with HTML5, CSS3, and JavaScript*. Apress, December, 15 2010.

(5) Flickenger, R. *How to Accelerate Your Internet: A Practical Guide to Bandwidth Management and Optimisation Using Open Source Software*. INASP, 2006.

(6) De Lucia, A., Ferrucci, F., Tortora, G., Tucci, M. *Emerging Methods, Technologies and Process Management in Software Engineering*. Wiley-IEEE Computer Society Pr, February 25, 2008.

(7) Shelly, G., B., Frydenberg, M. *Web 2.0: Concepts and Applications*. Cengage Learning, March 3, 2010.

(8) MacDonald, M., Freeman, A. *Pro ASP.NET 4 in C# 2010*. Apress, June 30, 2010.

(9) Williams, H., E., Lane, D. *Web Database Applications with PHP and MySQL*. O'Reilly Media, May 16, 2004.

(10) Joshi Joshi. *HTML5 Programming for ASP.NET Developers*. Apress, November 7, 2012.

(11) Stephens, S., Kruckenberg, M., Bouman, R. *Mysql 5.1 Cluster Dba Certification Study Guide*. Lulu Enterprises, UK Ltd, November 30, 2007.

(12) Dino Esposito, *Microsoft ASP.NET and AJAX: Architecting Web Applications: Architecting Web Applications*. Microsoft Press, April 15, 2009.

(13) Jeff Johnson. *Gui Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann, March 31, 2000.

(14) Buyya, R., Broberg, J., Goscinski, A. *Cloud Computing: Principles and Paradigms,* John Wiley And Sons, March 15, 2011.

(15) Olsson M., Widell, N. *A Brand New Offline Web, Identifying and Mitigating Hurdles for Developers*. Retrieved from http://www.w3.org/2011/web-apps-ws/Papers.html, November 5th, 2011.

(16) Piccoli, G. *Essentials of Information Systems for Managers*. John Wiley And Sons, January 24, 2012.

(17) Jones, T., Dewing, C. *Future Agenda*. Infinite Ideas, December 18, 2010.

(18) Nahari, H., Krutz, R.,L. *Web Commerce Security Design and Development*. John Wiley And Sons, April 26, 2011.

(19) Wildermuth, S.,Blomsma, M.,Wightman, J. *MCTS Self-Paced Training Kit (Exam 70-561): Microsoft .NET Framework 3.5 - ADO.NET Application Development*. Microsoft Press, March 25, 2009.

(20) Steve Johnson. *Microsoft Office Access 2007 on Demand*. Que Publishing, January 9, 2007.

(21) Matthew Crowley. *Pro Internet Explorer 8 and 9 Development: Developing Powerful Applications Development*. Apress, November 24, 2010.

(22) Stephanie Sammartino McPherson. *Tim Berners-Lee: Inventor of the World Wide Web*. Lerner Publishing Group, September 1, 2009.

(23) Retrieved from http://www.w3schools.com/htmldom/default.asp

(24) *Prototyping Responsive HTML5 Web Apps*.
Retrieved from http://www.youtube.com/watch?v=V2EjipWZ7co. MarakanaTechTV, December 2011.

(25) Dafydd Stuttard, Marcus Pinto. *The Web Application Hacker's Handbook*. Wiley, September 27, 2011.

(26) Keith, J., Sambells, J. *DOM Scripting: Web Design with JavaScript and the Document Object Model* (2nd edition). Friends of ED, December 27, 2010.

(27) Rob Crowther. *Hello! HTML5 & CSS3: A user-friendly reference guide*. Manning Publications, October 29, 2012.

(28) Retrieved from http://www.youtube.com.

(29) Adam Freeman. *The Definitive Guide to HTML5*. Apress, December 14, 2011.

(30) Kyrnin, J., Hudson, C., Leadbetter, T. *The HTML5 Developer's Collection*. Addison-Wesley Professional, December 29, 2011.

(31) Wilton, P., McPeak, J. *Beginning JavaScript* (4th edition). Wrox, October 26, 2009.

(32) Rich Page. *Website Optimization: An Hour a Day*. Sybex, May 8, 2012.

(33) Zabir, O., A. *Building a Web 2.0 Portal with ASP.NET 3.5*. O'Reilly Media, January 11, 2008.

(34) Larry, D., Lars, D. *Digital Forensics for Legal Professionals*. Syngress, September 16, 2011.

(35) Mackin, J., C., Russel, C. *Windows Essential Business Server 2008 Administrator's Companion*. Microsoft Press, May 13, 2009.

(36) Greg Barish. *Building Scalable and High-Performance Java Web Applications: Using J2EE Technology*. Addison-Wesley Professional, December 27, 2001.

(37) Retrieved from http://caniuse.com/indexeddb.

(38) Scott Preston. *Learn HTML5 and JavaScript for iOS*. Apress, May 02, 2012.

(39) Adrian Kosmaczewski. *Mobile JavaScript Application Development*. O'Reilly Media, June 27, 2012.

(40) Steven Cheng. *Odata Programming Cookbook for .Net Developers*. Packt Publishing, July 25, 2012.

(41) Jesse Cravens, Jeff Burtoft. *HTML5 Hacks*. O'Reilly Media, December 5, 2012.

(42) Jonathan Stark, Brian Jepson. *Building Android Apps with HTML, CSS, and JavaScript* (2nd edition). O'Reilly Media, January 30, 2012.

(43) Carsten Eilers. *HTML5 Security*. Developer Press, October 11, 2012.

(44) McDaniel, A. *HTML5: Your visual blueprint for designing rich web pages and applications*. Visual, November 8, 2011.

(45) Nicholas C. Zakas. *Professional: JavaScript for Web Developers* (3rd edition). Wrox, January 18, 2012.

(46) Daniel Mohl. *Building Web, Cloud, and Mobile Solutions With F#*. O'Reilly Media, December 28, 2012.

(47) Kroeger, R. *Cache Pattern for Offline Web Applications*. Retrieved from https://dl.google.com/io/2009/pres/W_0145_CachePatternforOfflineWebApplications.pdf. June 27, 2009.

(48) Steve Souders. *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly Media, June 17, 2009.

(49) Shelley Powers. *Learning Node*. O'Reilly Media, October 10, 2012.

(50) Retrieved from http://www.websocket.org/aboutwebsocket.html.

(51) James Pearce. *Professional Mobile Web Development with WordPress, Joomla and Drupal*. Wrox, April 12, 2011.

(52) Christian Gross. *Ajax Patterns and Best Practices*. Apress, February 16, 2006.

(53) Jesse Liberty, Dan Hurwitz. *Programming .NET Windows Applications*. O'Reilly Media, November 4, 2003.

(54) Duane Wessels. *Web Caching*. O'Reilly Media, June 2001.

(55) Matthew Ellis. *ASP.Net Ajax Programming Tricks*. Magma Interactive, LLC, October 30, 2007.

(56) Dinesh C. Verma. *Content Distribution Networks: An Engineering Approach* (1st edition). Wiley-Interscience, December 15, 2001.

(57) Sasha Goldshtein, Dima Zurbalev, Ido Flatow. *Pro .NET Performance: Optimize Your C# Applications*. Apress, September 12, 2012.

(58) Sushil Jajodia, Larry Kerschberg. *Advanced Transaction Models and Architectures*. Springer, August 31, 1997.

(59) Mark Grand. *Java Enterprise Design Patterns: Patterns in Java*. Addison-Wesley Professional, November 15, 2002.

(60) Andrew B. King. *Website Optimization*. O'Reilly Media, July 15, 2008.

(61) Joe Lewis, Meitar Moscovitz. *AdvancED CSS*. Friends of ED, July 29, 2009.

(62) Schaefer, K., Cochran, J., Forsyth, S., Everest, M., Glendenning, D. *Professional IIS 7*. Wrox, March 10, 2008.

(63) William R. Stanek. *Internet Information Services (IIS) 7.0 Administrator's Pocket Consultant*. Microsoft Press, December 29, 2007.

(64) Athena Vakali, George Pallis. *Web Data Management Practices: Emerging Techniques And Technologies*. IGI Global, August 29, 2006.

(65) Michael Brown. *MVVM Unleashed*. Sams Publishing, 2013.

(66) John Ciliberti. *ASP.NET MVC 4 Recipes: A Problem-Solution Approach*. Apress, February 20, 2013.

(67) Sergey Mavrody. *Sergey's HTML5 & CSS3: Quick Reference* (2nd edition). Belisso, January 9, 2012.

(68) S. V. Nagaraj. *Web Caching and Its Applications*. Springer, June 1, 2004.

(69) Peter Wainwright. *Pro Apache* (3rd edition). Apress, January 22, 2004.

(70) Neal Ford. *The Productive Programmer*. O'Reilly Media, July 10, 2008.

(71) Christian Gross. *Beginning C# 2008: From Novice to Professional* (2nd edition). Apress, September 16, 2008.

(72) J. D. Meier, Microsoft Corporation. *Improving Web Application Security: Threats and Countermeasures*. Microsoft Press, September 2, 2003.

(73) Michael Dory, Adam Parrish, Brendan Berg. *Introduction to Tornado*. O'Reilly Media, March 28, 2012.

(74) Ec-Council, Course Technology. *Web Applications and Data Servers*. Cengage Learning, September 24, 2009.

(75) Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, March 3, 2012.

(76) Fowler, M. *Domain Specific Languages*. Addison-Wesley Professional, October 3, 2010.

(77) Shelley Powers. *JavaScript Cookbook*. O'Reilly Media, July 26, 2010.

(78) Retrieved from http://www.websocket.org/quantum.html.

(79) Flanagan, D. *JavaScript: The Definitive Guide* (6th edition). O'Reilly Media, May 10, 2011.

(80) *Mobile Web Apps vs. Mobile Native Apps: How to Make the Right Choice* (White paper), Retrieved from http://www.lionbridge.com/files/2012/11/Lionbridge-WP_MobileApps2.pdf, Lionbridge, November, 2012.

(81) Hill, D., Webster, B., Jezierski, E, A., Vasireddy, S., Al-Sabt, M., Rasmusson, J., Gale, P., Slater, P. *Microsoft Patterns and Practices: Smart Client Application Architecture and Design Guide*. Microsoft Press, 2004.

(82) Williams, James, L. *Learning HTML5 Game Programming*. Addison-Wesley Professional, October 5, 2011.

(83) Dominic, D. *Enterprise Software Architecture and Design*. Wiley-IEEE Computer Society Pr, February 28, 2012.

(84) Singh S. K. *Database Systems: Concepts, Design and Applications*. Prentice Hall, August 10, 2009.

(85) Mano Paul. *Official (ISC)2 Guide to the CSSLP*. CRC Press, June 13, 2011.

(86) Charles Garrod. *Putting the scalability Into Database Scalability Services*. ProQuest, UMI Dissertation Publishing, September 10, 2011.

(87) Jeremiah Grossman. *Cross-site Scripting Attacks*. Syngress, May 15, 2007.

(88) Mark Burnett. *Hacking the Code: Auditor's Guide to Writing Secure Code for the Web* Syngress, May 10, 2004.

(89) Nilachala, P. *Developing Offline Web Applications using HTML5*. Retrieved from http://www.tcs.com/SiteCollectionDocuments/White Papers/TEG_Whitepaper_Developing_Offline_Web_Application_Using_HTML5_0212-1.pdf (White Paper), February 2012.

(90) Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, December 12, 2002.

(91) Javier Lopez, Bernhard Hammerli. *Critical Information Infrastructures Security*: Second International Workshop, CRITIS 2007, Benalmadena-Costa, Spain. October 3-5, 2007.

(92) Don Gosselin. *JavaScript* (4th edition). Course Technology, September 7, 2007.

(93) Martin, R, C. *Best Practices for Mobile Application Architectures*. Retrieved from http://i.i.com.com/cnwk.1d/html/itp/iAnywhere_Smart_Client_Architectures_MobileDev.pdf. May 8, 2008.

(94) Martin, R, C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, August 11, 2008.

(95) Kyrnin, J. *Sams Teach Yourself HTML5 Mobile Application Development in 24 Hours*. Sams Publishing, November 25, 2011.

(96) Applying Method Chaining: First Class Thoughts, Retrieved from http://firstclassthoughts.co.uk/java/method_chaining.html. 2011.

(97) Turner, D., Fossi, M., Johnson, E., ,Mack, T., Blackbird, J., Entwisle, S., Low, M, K., McKinney, D., Wueest, C. *Symantec Internet Security Threat Report*: Trends for July-December 2007. Volume 13, published April 2008.

(98) Christine Ennew, Nigel Waite. *Financial Services Marketing*. Routledge, November 13, 2006.

(99) Daniel Guagnin, Leon Hempel, Carla Ilten, Inga Kroener, Daniel Neyland. *Managing Privacy Through Accountability*. Palgrave Macmillan, September 18, 2012.

(100) Committee on Authentication Technologies and Their Privacy Implications, National Research Council. *Who Goes There?: Authentication Through the Lens of Privacy*. National Academies Press, March 25, 2003.