

Università degli Studi *“Roma Tre”*

Dipartimento di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

***X-Learning***

Laureando

Alessandro Rastelli

Relatore Correlatori

Prof. Alberto Paoluzzi Dott. Enrico Marino, Dott. Federico Spini

Anno Accademico 2014/2015

*Dedicated to my families*

Acknowledgements

Firstly, we would like to express our sincere gratitude to our advisors, Enrico Marino and Federico Spini, for the continuous support of my project, for their patience, motivation, and immense knowledge. Besides our advisors, we would like to thank Prof. Alberto Paoluzzi for his supervision and the opportunity that he oﬀered to us.

**Introduction**

**Introduzione**

**Chapter 1**

**Massive open online course**

This chapter describes MOOC’s world.

In the first section a MOOCs overview is provided: what MOOCs are, what they are for and how do they work. In the second section it’s reported a classification of currently avalaible MOOCs.

* 1. MOOC overview

This section consists of an overview about a massive open online course (MOOC). A MOOC is an online course aimed at unlimited participation and open access via the web. In addition to traditional course materials such as filmed lectures, readings, and problem sets, many MOOCs provide interactive user forums to support community interactions between students, professors, and teaching assistants (TAs). MOOCs are a recent and widely researched development in distance education which were first introduced in 2008 and emerged as a popular mode of learning in 2012.[Pappano, Laura. "The Year of the MOOC". The New York Times. Retrieved 18 April 2014.][Lewin, Tamar (20 February 2013). "Universities Abroad Join Partnerships on the Web". New York Times. Retrieved 6 March 2013.]. Early MOOCs often emphasized open-access features, such as open licensing of content, structure and learning goals, to promote the reuse and remixing of resources. Some later MOOCs use closed licenses for their course materials while maintaining free access[[1]](#footnote-2) for students. [Wiley, David. "The MOOC Misnomer". July 2012] [Cheverie, Joan. "MOOCs an Intellectual Property: Ownership and Use Rights". Retrieved 18 April 2013.] [David F Carr (20 August 2013). "Udacity hedges on open licensing for MOOCs". Information Week. Retrieved 21 August 2013.][P. Adamopoulos, "What Makes a Great MOOC? An Interdisciplinary Analysis of Student Retention in Online Courses", ICIS 2013 Proceedings (2013) pp. 1–21 in AIS Electronic Library (AISeL)].

The first MOOCs emerged from the open educational resources (OER) movement. The term MOOC was coined in 2008 by Dave Cormier of the University of Prince Edward Island in response to a course called Connectivism and Connective Knowledge (also known as CCK08). CCK08, which was led by George Siemens of Athabasca University and Stephen Downes of the National Research Council, consisted of 25 tuition-paying students in Extended Education at the University of Manitoba.

Alongside the development of these open courses, other E-learning platforms emerged - such as Khan Academy, Peer-to-Peer University (P2PU), Udemy and ALISON - which are viewed as similar to MOOCs and work outside the university system or emphasize individual self-paced lessons.[Yuan, Li, and Stephen Powell. MOOCs and Open Education: Implications for Higher Education White Paper. University of Bolton: CETIS, 2013. pp. 7–8.][ "What You Need to Know About MOOCs". Chronicle of Higher Education. Retrieved 14 March 2013.][ "Open Education for a global economy".][ Booker, Ellis (30 January 2013). "Early MOOC Takes A Different Path". Information Week. Retrieved 25 July 2013.][Bornstein, David (11 July 2012). "Open Education For A Global Economy". New York Times. Retrieved 25 July 2013.].

According to The New York Times, 2012 became "the year of the MOOC" as several well-financed providers, associated with top universities, emerged, including Coursera, Udacity

In January 2013, Udacity launched its first MOOCs-for-credit, in collaboration with San Jose State University. In May 2013 the company announced the first entirely MOOC-based master's degree, a collaboration between Udacity, AT&T and the Georgia Institute of Technology, costing $7,000, a fraction of its normal tuition.["Georgia Tech, Udacity Shock Higher Ed With $7,000 Degree". Forbes. 2012-04-18. Retrieved 2013-05-30.]

* 1. MOOC classification

Coursera is a platform MOOC Established by Professors of Computer Science Andrew Ng and Daphne Koller Stanford University. This platform have more than 150 International University Partners of Which anche the University of Rome La Sapienza and the Bocconi University of Milan, to allow the platform to offer courses in many different languages ​​Among Which including Italian. All courses are accessible Coursera For free.

For those interested in some courses are given the opportunity to obtain, at the end of course, a "Verified certificate" a certificate issued by the university providing the course. The certificates are paid and have a variable cost.

Even Coursera courses generally begin at a precise date and have an average duration of a few weeks. They are divided into modules which are publishe online weekly. Forms can be made either by video contributions from both textual contributions. In the videos. Furthermore, the speed of the speech, can be adjusted and in some cases they are provided with subtitles in many different languages.

For most of the courses are provided for the quiz intermediate, generally relating to a group of modules at a time, and a final test of the course.

Each course has a discussion forums (divided into sections), a showcase for the publication of the ads, a community that helps you get to know other students of the course and wiki, useful as external support to complete course units. Unlike other platforms MOOC presented on this page, Coursera provides for the acceptance of an honor code before you can start your course.

Udacity was born from an experiment at Stanford University. The portal offers scientific, of design and business courses, divided by the various levels. The courses are provided to non-traditional teaching methods.

The certificates are released at the end of the course if the candidate has to prove his identity and has attained the objectives of the course without external aid. Some udacity's courses are fee.

The Udacity’s courses are all self-paced they are organized primarily according modules. This modules have textual and video contributions, but also offer activities and short interviews with professors. In some cases they are provided English subtitles for the videos to which you can also adjust the speed of the speech.

Also the courses offer by Udacity have a showcase for ads, forum, wiki and community. For most of the courses are provided for the quiz intermediate, generally relating to a group of modules at a time, and a final test of the course. Unlike other platforms, on Udacity, quizzes can be repeated without this affecting the final grade.



Udemy is platform or marketplace for online learning that allow anyone to publish video online courses. There is a version of the site in Italian but the courses are in English and in other languages ​​such as French, Spanish and Chinese. At the end of all courses released a certificate. Udemy provides a platform for experts of any kind to create courses which can be offered to the public, either at no charge or for a tuition fee, but in some cases is possible to see some of the first lessons for free.

The courses Udemy are all self-paced. They are organized in modules but they are not released weekly. Most of the video is not provided with subtitles. They are, moreover, all the most common tools of other platforms such as forums, communities, wikis and chat rooms, however there is a great and original function for taking notes during lessons.

Udemy has made a special effort to attract corporate trainers seeking to create coursework for employees of their company.[Carr, David F. Udemy Comes To Corporate Training Information Week. April 16, 2013]

**Chapter 2**

**Enabling Technologies**

This chapter describes X-Learning enabling technologies.

The first three sections concern server-side technologies: MongoDB, NodeJS and Loopback by Strongloop (an IBM company). MongoDB is a NoSQL document-oriented database management system; NodeJS is an event-driven framework to handle Javascript server sides; Loopback is a NodeJS based framework created to use and edit set of APIs.

The fourth, fifth and sixth sections are related to client-side technologies: HTML5, Web Components and Polymer-Project by Google. HTML5 is a markup language aimed at web pages structuring; Web Components are a set of standards that allow for the creation of reusable widget and components in web documents; Polymer-Project provides a thin layer of API on top of Web Components and several powerful features, such as custom events, delegation, mixins, accessors and component life-cycle functions, to facilitate the creation of Web Components.

* 1. HTML 5

This section provides an overview of HTML5.

HTML5 is the latest version of Hypertext Markup Language, the code



Figure 2.1: Server and client sides enabling technologies

that describes web pages. There are actually three kinds of code: HTML, which provides the structure; Cascading Style Sheets (CSS), which take care of presentation; and JavaScript, which makes things happen.

HTML5 has been designed to deliver almost everything it is possible to do online without requiring additional software such as browser plugins. It does everything, from animation to apps, music to movies, and can also be used to build complicated applications that run in browsers.

Moreover, HTML5 isn’t proprietary, so it is completely free. It’s also a cross-platform standard, which means it doesn’t care whether the device is a tablet or a smartphone, a netbook, notebook or ultrabook or a Smart TV: if the browser supports HTML5, it should work flawlessly.

While some features of HTML5 are often compared to Adobe Flash, the two technologies are very different. Both include features for playing audio and video within web pages, and for using Scalable Vector Graphics. HTML5, on its own, cannot be used for animation or interactivity, it must be

supplemented with CSS3 or JavaScript. There are many Flash capabilities that have no direct counterpart in HTML5. See Comparison of HTML5 and Flash.

Figure 2.2: Html5 Responsive

Although HTML5 has been well known among web developers for years, its interactive capabilities became a topic of mainstream media around April 2010, after Apple Inc’s then-CEO Steve Jobs issued a public letter entitled “Thoughts on Flash” where he concluded that “Flash is no longer necessary to watch video or consume any kind of web content” and that “new open standards created in the mobile era, such as HTML5, will win”.This sparked a debate in web development circles where some suggested that while HTML5 provides enhanced functionality, developers must consider the varying browser support of the different parts of the standard as well as other functionality differences between HTML5 and Flash. In early November 2011, Adobe announced that it would discontinue development of Flash for mobile devices and reorient its efforts in developing tools using HTML5.

* 1. Web Components

This section provides an overview of Web Components.

Web Components are a set of standards currently being produced by Google engineers as a W3C specification that allows for the creation of reusable widgets or components in web documents and web applications. The intention behind them is to bring component-based software engineering to the World Wide Web. The components model allows for encapsulation and interoperability of individual HTML elements.

Support for Web Components is present in some WebKit-based browsers like Google Chrome and Opera and is in Mozilla Firefox (requires a manual configuration change). Microsoft’s Internet Explorer has not implemented any Web Components specifications yet.[1] Backwards compatibility with older browsers is implemented using JavaScript-based polyfills.[35]

Web Components consist of 4 main elements which can be used separately or all together:

* + - Custom Elements

Custom Elements allow authors to define their own custom HTML elements. Authors associate JavaScript code with custom tag names, and then use those custom tag names as they would any standard tag. Custom elements are still elements. It is possible to create, use,

manipulate, and compose them just as easily as any standard <div> or <span> today.[8]

* + - Shadow DOM

Shadow DOM addresses the lack of true DOM tree encapsulation when building components. With Shadow DOM, elements can get a new kind of node associated with them. This new kind of node is called a shadow root. An element that has a “shadow root” associated with it is called a “shadow host”. The content of a shadow host isn’t rendered; the content of the shadow root is rendered instead. Shadow DOM allows a single node to express three subtrees: light DOM, shadow DOM, and composed DOM. Together, the light DOM and shadow DOM are referred to as the logical DOM. This is the DOM that the developer interacts with. The composed DOM is what the browser sees and uses to render the pixels on the screen.[9]

*Structure of a Shadow DOM* An element that has a shadow root asso- ciated with it is called shadow host. The shadow root can be treated as an ordinary DOM element, so it is possible to append arbitrary nodes to it. With Shadow DOM, all markup and CSS are scoped to the host element. In other words, CSS styles defined inside a Shadow Root won’t aﬀect its parent document, CSS styles defined outside the Shadow Root won’t aﬀect the main page.

* + - HTML Import

This webcomponents.js repository contains a JavaScript polyfill for the HTML Imports specification. HTML Imports are a way to include and reuse HTML documents in other HTML documents. As <script> tags let authors include external JavaScript in their pages, imports let authors load full HTML resources. In particular, imports let authors include Custom Element definitions from external URLs.

* + - Templates

This specification describes a method for declaring inert DOM subtrees in HTML and manipulating them to instantiate document fragments with identical contents.

* 1. Polymer

This section there will be an overview of Polymer. Polymer provides a thin layer of API on top of Web Components and several powerful features, such as custom events and delegation, mixins, accessors and component life- cycle functions, to facilitate the creation of Web Components. Polymer does this by:

* + - Allowing to create Custom Elements with user-defined naming schemes. These custom elements can then be distributed across the network and used by others with HTML Imports
    - Allowing each custom element to have its own template accompanied by styles and behavior required to use that element
    - Providing a suite of ready-made UI and non-UI elements to be used and extended in projects

The elements collection of Polymer is divided into more sections:

* + - Core Elements — These are a set of visual and non-visual elements designed to work with the layout, user interaction, selection, and scaf- folding applications.
    - Paper Elements — Implement the material design philosophy launched by Google recently at Google I/O 2014, and these include everything from a simple button to a dialog box with neat visual eﬀects.
    - Iron Elements — A set of visual and non-visual utility elements. It includes elements for working with layout, user input, selection, and scaﬀolding apps.
    - Gold Elements — The gold elements are built for e-commerce use-cases like checkout flows.
    - Neon Elements — Neon elements implement special eﬀects.
    - Platinum Elements — Elements to turn web pages into a true webapp, with push, oﬄine, and more.
    - Molecules — Molecules are elements that wrap other javascript li- braries.

Figure 2.3: Polymer Architecture

Web components standards provide the needed primitives to build new components. It is possible to build custom elements using these primitives, but it can be a lot of work.

The Polymer library provides a declarative syntax that makes it simpler to define custom elements. Furthermore, it adds features like templating, two-way data binding and property observation to help developers build powerful, reusable elements with less code.

Custom elements. If users don’t want to write their own elements, there are a number of elements built with Polymer that it is possible to drop straight into existing pages. These elements depend on the Polymer library, but they can be used without using Polymer directly, as well.[6]

Polymer is one of the first implementations of a user interface library built upon the Web Components standard. Web Components are not fully supported by browsers, but they provide a polyfill library, webcomponents.js, that provides enough functionality to support Web Components and Poly- mer.

Web Components standard is the result of the evolution of user interface libraries over the past decade, finally reaching the goal of separating HTML, CSS and JavaScript and running HTML through W3C validators. For exam- ple, looking at a .css file, it is possbile to easily determine which selectors are actually used in HTML and especially programmatically used in JavaScript. Similarly, it is easy to organize JavaScript code so that everything could be reused eﬃciently on multiple pages.[23]

* 1. NodeJS

This section provides an overview of NodeJs.

Node.js is an open source, cross-platform runtime environment for server- side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows,

Linux, FreeBSD, NonStop,IBM AIX, IBM System z and IBM i. Its work is hosted and supported by the Node.js Foundation,a Collaborative Project at Linux Foundation.

Node.js provides an event-driven architecture and a non-blocking I/O API that optimizes an application’s throughput and scalability. These tech- nologies are commonly used for real-time web applications.

Node.js uses the Google V8 JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript. Node.js contains a built-in library to allow applications to act as a Web server without software such as Apache HTTP Server, Nginx or IIS.

Node.js allows the creation of web servers and networking tools, using JavaScript and a collection of “modules” that handle various core functional- ity. Modules handle file system I/O, networking (HTTP, TCP, UDP, DNS, or TLS/SSL), binary data (buﬀers), cryptography functions, data streams , and other core functions. Node’s modules have a simple and elegant API, reducing the complexity of writing server applications.

Frameworks can be used to accelerate the development of applications, and common frameworks are Express.js, Socket.IO and Connect. Node.js applications can run on Microsoft Windows, Unix, NonStop and Mac OS X servers. Node.js applications can alternatively be written with CoﬀeeScript (an alternative form of JavaScript), Dart or Microsoft TypeScript (strongly typed forms of JavaScript), or any language that can compile to JavaScript. Node.js is primarily used to build network programs such as web servers, making it similar to PHP and Python. The biggest diﬀerence between PHP and Node.js is that PHP is a blocking language (commands execute only after the previous command has completed), while Node.js is a non-blocking language (commands execute in parallel, and use callbacks to signal comple-

tion).

Node.js brings event-driven programming to web servers, enabling de velopment of fast web servers in JavaScript. Developers can create highly scalable servers without using threading, by using a simplified model of event- driven programming that uses callbacks to signal the completion of a task. Node.js was created because concurrency is diﬃcult in many server-side pro- gramming languages, and often leads to poor performance. Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming.

* 1. MongoDB

This section provides an overview of MongoDB.

MongoDB is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Aﬀero General Public License and the Apache License, MongoDB is free and open-source software.

MongoDB was created by Dwight Merriman and Eliot Horowitz, who had encountered development and scalability issues with traditional relational database approaches while building Web applications at DoubleClick, an Internet advertising company that is now owned by Google Inc. According to Merriman, the name of the database was derived from the word humongous to represent the idea of supporting large amounts of data. Merriman and Horowitz helped form 10Gen Inc. in 2007 to commercialize MongoDB and related software. The company was renamed MongoDB Inc. in 2013.

The database was released to open source in 2009 and is available under the terms of the Free Software Foundation’s GNU AGPL Version 3.0 com- mercial license. At the time of this writing, among other users, the insurance company MetLife is using MongoDB for customer service applications, the

website Craigslist is using it for archiving data, the CERN physics lab is using it for data aggregation and discovery and the The New York Times newspaper is using MongoDB to support a form-building application for photo submissions.

Figure 2.4: MongoDB Architecture

* 1. **StrongLoop LoopBack**

This section provides an overview of LoopBack.

Built on top of the open source LoopBack framework, the StrongLoop API Platform is the first end-to-end platform for the full API lifecycle that allows to visually develop REST APIs in Node and get them connected to new and legacy data. In addition, the API Platform features built-in mBaaS1 Mobile Backend as a service (MBaaS), also known as “backend as a service” (BaaS),

is a model for providing web and mobile app developers with a way to link their ap-

plications to backend cloud storage and APIs exposed by back end applications while

features like push and oﬄine sync, plus graphical tools with DevOps features for clustering, profiling and monitoring Node apps.

LoopBack generates model API from the models schemas, to let CRUD operations on models.

LoopBack models automatically have a standard set of HTTP endpoints that provide REST APIs for create, read, update, and delete (CRUD) oper- ations on model data:

* + - **POST /Model** — Create a new instance of the model and persist it into the data source.
    - **GET /Model** — Find all instances of the model matched by filter from the data source.
    - **PUT /Model** — Update an existing model instance or insert a new one into the data source.
    - **PUT /Model/id** — Update attributes for a model instance and per- sist it into the data source.
    - **GET /Model/id** — Find a model instance by id from the data source.
    - **DELETE /Model/id** — Delete a model instance by id from the data source.
    - **GET /Model/count** — Count instances of the model matched by where from the data source.
    - **GET /Model/findOne** Find first instance of the model matched by filter from the data source.
    - **POST /Model/update** — Update instances of the model matched by where from that data source.

also providing features such as user management, push notifications, and integration with social networking services.

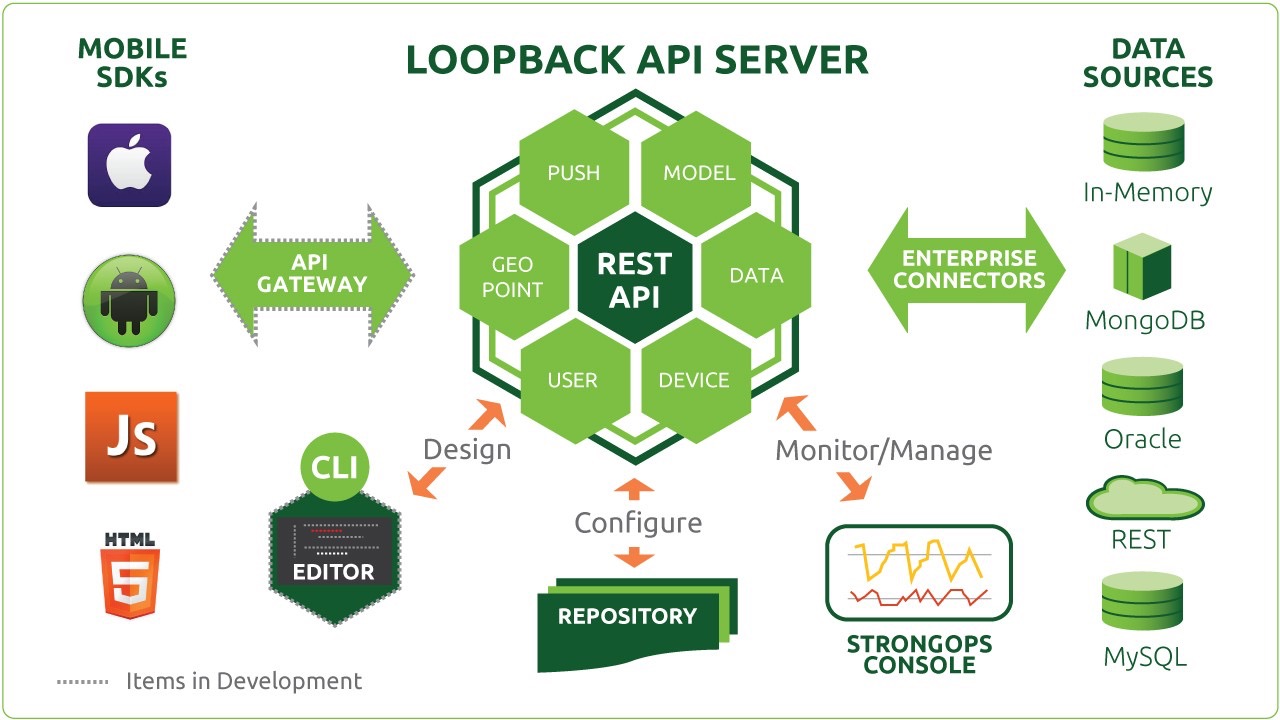


Figure 2.5: LoopBack Architecture

A LoopBack model represents data in backend systems such as databases, and by default has both Node and REST APIs. Additionally, developer can add functionality such as validation rules and business logic to models. Every LoopBack application has a set of predefined built-in models such as User, Role, and Application. Developer can extend built-in models to suit application’s needs.

The model JSON file defines models, relations between models, and ac- cess to models.

"http": {"path": "/foo/mypath"}

}

Where:

* + - “name”: Name of the model.
    - “description”: Optional description of the model.
    - “base”: Name of another model that this model extends. The model will “inherit” properties and methods of the base model.
    - “IdInjection”: Whether to automatically add an id property to the model:
      * true - id property is added to the model automatically. This is the default.
      * false - id property is not added to the model.
    - “strict”: Specifies whether the model accepts only predefined prop- erties or not. One of:
      * true - Only properties defined in the model are accepted. Used to ensure that the model accepts only predefined properties.
      * false - The model is an open model and accepts all properties, including ones not predefined in the model. This mode is useful to store free-form JSON data to a schema-less database such as MongoDB.
      * validate - The unknown properties will be reported as valida- tion errors.
      * throw - Throws an exception if properties not defined for the model are used in an operation.
      * undefined - Defaults to false unless the data source is backed by a relational database such as Oracle or MySQL.
    - “options”: JSON object that specifies model options.
    - “properties”: JSON object that specifies the properties in the model.
    - “relations”: Object containing relation names and relation defini- tions.
    - “acls”: Set of ACL specifications that describes access control for the model.

The API can be extended: the developer can add remote functions to models or add hooks to existing API to add custom behavior before and/or after the API handler (to pre-process the re-quest and/or post-process the response). The resulting API is RESTful, cookie free, signed by authentica- tion token. By default, applications have a built-in model that represents a user, with properties username, email and password and role for authentica- tion and authorization. Loopback also introduces an indirection layer that allows to choose from almost all particular DBMS to be used.

In Chapter Two the technologies used for developing this work, have been described. Each technology has been described in realtion to its function and its use in the project.

**Chapter 3**

**Single Page Application**

This chapter provides an overview of Single Page Application develop- ment pattern. The first section analyzes the pattern in its totality and pro- vides general and specific definitions. The second section presents pros and cons of the use of Single Page Application, such as SEO problems and speed of load. The third section explains technical functioning of the pattern.

* 1. Single Page Application

A single-page application (SPA) is a web application or web site that fits on a single web page with the goal of providing a more fluid user expe- rience akin to a desktop application. In a SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load,[31] or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does control transfer to another page, although modern web technologies (such as those included in the HTML5 pushState() API) can provide the perception and navigability of separate logical pages in the application. Interaction with the single page application often involves dynamic communication with the web server behind the scenes.

SPAs use AJAX and HTML5 to create fluid and responsive Web apps, without constant page reloads. However, this means much of the work hap- pens on the client side, in JavaScript. For the traditional ASP.NET devel- oper, it can be diﬃcult to make the leap. Luckily, there are many open source JavaScript frameworks that make it easier to create SPAs.

In a traditional Web app, every time the app calls the server, the server renders a new HTML page. This triggers a page refresh in the browser. In an SPA, after the first page loads, all interaction with the server happens through AJAX calls. These AJAX calls return data—not markup—usually in JSON format. The app uses the JSON data to update the page dynami- cally, without reloading the page.

* 1. Single Page Application: pros and cons

One benefit of SPAs is obvious: Applications are more fluid and respon- sive, without the jarring eﬀect of reloading and re-rendering the page. An- other benefit is provided by ending the app data as JSON creates a separation between the presentation (HTML markup) and application logic (AJAX re- quests plus JSON responses). This separation makes it easier to design and evolve each layer. In a well-architected SPA, it is possbile to change the HTML markup without touching the code that implements the application logic. In a pure SPA, all UI interaction occurs on the client side, through JavaScript and CSS. After the initial page load, the server acts purely as a service layer. The client just needs to know what HTTP requests to send. It doesn’t care how the server implements things on the back end. With this architecture, the client and the service are independent. It is possbile to replace the entire back end that runs the service, and as long as the API doesn’t change, the client won’t break. The reverse is also true.

Because the SPA is an evolution away from the stateless page-redraw model that browsers were originally designed for, some new challenges have

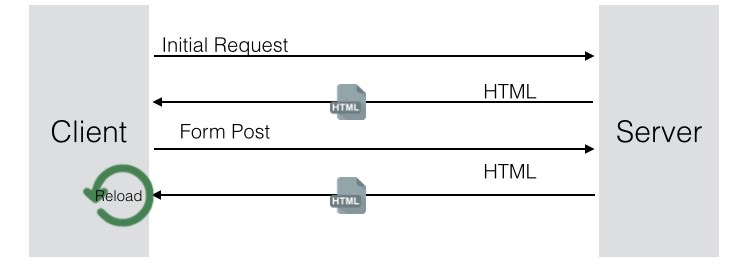


Figure 3.1: The Traditional Page Lifecycle

Figure 3.2: The SPA Lifecycle

emerged. Each of these problems has an eﬀective solution with:

* + - Client-side JavaScript libraries addressing various issues.
    - Server-side web frameworks that specialize in the SPA model.
    - The evolution of browsers and the HTML5 specification aimed at the SPA model.
    1. **Search engine optimization (SEO)**

Because of the lack of JavaScript execution on crawlers of some popu- lar Web search engines, SEO (Search engine optimization) has historically presented a problem for public facing websites wishing to adopt the SPA model. Google currently crawls URLs containing hash fragments starting with #!. This allows the use of hash fragments within the single URL of an SPA. Special behavior must be implemented by the SPA site to allow extraction of relevant metadata by the search engine’s crawler. For search engines that do not support this URL hash scheme, the hashed URLs of the SPA remain invisible. Alternatively, applications may render the first page load on the server and subsequent page updates on the client. This is traditionally diﬃcult, because the rendering code might need to be written in a diﬀerent language or framework on the server and in the client. Using logic-less templates, cross-compiling from one language to another, or us- ing the same language on the server and the client may help to increase the amount of code that can be shared. Because SEO compatibility is not trivial in SPAs, it’s worth noting that SPAs are commonly not used in a context where search engine indexing is either a requirement, or desirable. Use cases include applications that surface private data hidden behind an authentica- tion system. In the cases where these applications are consumer products, often a classic page redraw model is used for the applications landing page and marketing site, which provides enough meta data for the application to appear as a hit in a search engine query. Blogs, support forums, and other traditional page redraw artifacts often sit around the SPA that can seed search engines with relevant terms. Another approach used by server-centric web frameworks like the Java-based ItsNat is to render any hypertext in the server using the same language and templating technology. In this approach, the server knows with precision the DOM state in the client, any big or small page update required is generated in the server, and transported by AJAX, the exact JavaScript code to bring the client page to the new state executing DOM methods. Developers can decide which page states must be crawlable by web spiders for SEO and be able to generate the required state in load time generating plain HTML instead of JavaScript. In case of the ItsNat

framework, this is automatic because ItsNat keeps the client DOM tree in the server as a Java W3C DOM tree; rendering of this DOM tree in the server generates plain HTML in load time and JavaScript DOM actions for AJAX requests. This duality is very important for SEO because develop- ers can build with the same Java code and pure HTML-based templating the desired DOM state in server; on page load time, conventional HTML is generated by ItsNat making this DOM state SEO-compatible. As of ver- sion 1.3, ItsNat provides a new stateless mode, client DOM is not kept in the server because, in stateless mode client, DOM state is partially or fully recon- structed in the server when processing any AJAX request based on required data sent by client informing of the current DOM state; the stateless mode may be also SEO-compatible because SEO compatibility happens in load time of the initial page not aﬀected by stateful or stateless modes. There are a couple of workarounds to make it look as though the web site is crawlable. Both involve creating separate HTML pages that mirror the content of the SPA. Server could create a HTML-based version of the site and deliver that to crawlers, or it’s possible to use a headless browser such as PhantomJS to run JavaScript application and output the resulting HTML. Both of these do require quite a bit of eﬀort, and can end up giving a maintenance headache for the large complex sites. There are also potential SEO pitfalls. If server- generated HTML is deemed to be too diﬀerent from the SPA content, then the site will be penalized. Running PhantomJS to output the HTML can slow down the response speed of the pages, which is something for which search engines – Google in particular – downgrades the rankings.

* + 1. **Client/Server code partitioning**

One way to increase the amount of code that can be shared between servers and clients is to use a logic-less template language like Mustache or Handlebars. Such templates can be rendered from diﬀerent host languages,

such as Ruby on the server and JavaScript in the client. However, merely sharing templates typically requires duplication of business logic used to choose the correct templates and populate them with data. Rendering from templates may have negative performance eﬀects when only updating a small portion of the page—such as the value of a text input within a large tem- plate. Replacing an entire template might also disturb a user’s selection or cursor position, where updating only the changed value might not. To avoid these problems, applications can use UI data bindings or granular DOM manipulation to only update the appropriate parts of the page instead of re-rendering entire templates [34].

* + 1. **Analytics**

Analytics tools such as Google Analytics rely heavily upon entire new pages loading in the browser, initiated by a URL change. SPAs don’t work this way. After the first page load, all subsequent page and content changes are handled internally by the application. So the browser never triggers a new page load, nothing gets added to the browser history, and the analytics package has no idea who’s doing what on the site.

It’s possible to add page load events to an SPA using the HTML5 history API; this will help integrate analytics. The diﬃculty comes in managing this and ensuring that everything is being tracked accurately – this involves checking for missing reports and double entries. Some frameworks provide open source analytics integrations addressing most of the major analytics providers. Developer should integrate them into the application and make sure that everything is working correctly, but there’s no need to do everything from scratch.

* + 1. **Speed of initial load**

Single Page Applications have a slower first page load than server-based applications. This is because the first load has to bring down the framework and the application code before rendering the required view as HTML in the browser. A server-based application just has to push out the required HTML to the browser, reducing the latency and download time.

There are some ways of speeding up the initial load of an SPA, such as a heavy approach to caching and lazy-loading modules when needed. But it’s not possible to get away from the fact that it needs to download the framework, at least some of the application code, and will most likely hit an API for data before displaying something in the browser.This is a “pay me now, or pay me later” trade-oﬀ scenario. The question of performance and wait-times remains a decision that the developer must make [34].

* 1. Single Page Application: how it works

An approach to implementing the single page pattern on today’s web that makes it easy to get it right. It’s based on three principles.

* + - Every view must have a real URL. On a normal web page, at any time I can grab what’s in my browser’s URL bar and either share it or bookmark it.
    - Every link must be a real link. On a normal web page, I can view a link’s destination by hovering it with my mouse; right-clicking produces a menu of link-specific options. So many otherwise-good pages try to implement their own link-like behavior by catching clicks on specific DOM nodes but then screw up corner case behaviors like middle-click.
    - It’s ok to be less awesome on old browsers. (Of course, whether this is actually true depends on site’s goals.) One tactic for old browsers

is just to fail gracefully: if the site still works but is just slower, that’s ok. For example, Gmail used to implement each of its buttons with a soup of DOM nodes to get the gradients and rounded corners to show on IE6; on today’s web, maybe it’s ok to just use some newer CSS for those eﬀects and allow the buttons to be square and flat on IE6.

Because of principle 1,the app must be capable of rendering any given view from scratch, because the URL load might be a freshly started browser that just loaded a bookmark. Because of principle 2, links must be imple- mented as plain old <a> tags with an href that points at the URL of the resulting view. [24]

* + 1. **Local Routing**

This section presents the operation and the importance of the local router to manage page requests made by the client. When the standard pattern is used HTTP GET request is triggered to require a new page: the browser itself sends the page request to the server, which responds by sending the page or performing the desired action. With the pattern SPA, the workflow is diﬀerent: the use of a client side router prevents the browser’s default behav- ior via the preventDefault method. The implementation of this pattern involves the following steps: listening to the path change via onChange function; for each event of this type, the request is handled by a mapping path or a handler, in which each path is associated with a function. In gen- eral, these functions have the task of retrieve the template and the content needed to fill it.

* + 1. **Server architecture**

#### Thin server architecture

A SPA moves logic from the server to the client. This results in the role of the web server evolving into a pure data API or web service. This architectural shift has, in some circles, been coined “Thin Server Architecture” to highlight that complexity has been moved from the server to the client, with the argument that this ultimately reduces overall complexity of the system.

#### Thick stateful server architecture

The server keeps the necessary state in memory of the client state of the page. In this way, when any request hits the server (usually user actions), the server sends the appropriate HTML and/or JavaScript with the concrete changes to bring the client to the new desired state (usually adding/delet- ing/updating a part of the client DOM). At the same time, the state in server is updated. Most of the logic is executed on the server, and HTML is usually also rendered on the server. In some ways, the server simulates a web browser, receiving events and performing delta changes in server state which are automatically propagated to client.

This approach needs more server memory and server processing, but the advantage is a simplified development model because a) the application is usually fully coded in the server, and b) data and UI state in the server are shared in the same memory space with no need for custom client/server communication bridges.

#### Thick stateless server architecture

This is a variant of the stateful server approach. The client page sends data representing its current state to the server, usually through AJAX re- quests. Using this data, the server is able to reconstruct the client state of the part of the page which needs to be modified and can generate the neces- sary data or code (for instance, as JSON or JavaScript), which is returned to the client to bring it to a new state, usually modifying the page DOM tree according to the client action which motivated the request.

This approach requires that more data be sent to the server and may require more computational resources per request to partially or fully recon- struct the client page state in the server. At the same time, this approach is more easily scalable because there is no per-client page data kept in the server and, therefore, AJAX requests can be dispatched to diﬀerent server nodes with no need for session data sharing or server aﬃnity. [34]

This chapter has provided an overview of Single Page Application develop- ment pattern. The first section has analyzed the pattern in its totality and has provided general and specific definitions. The second section has been presented pros and cons of the use of Single Page Application, such as SEO problems and speed of load. The third section has explained the technical functioning of the pattern.

**Chapter 4**

**X-Learning**

This chapter presents the core of the thesis project: X-Learning.

The first section provides a project’s overview, giving reasons of develop- ment, listing benefits and functions. The second section briefly explains of the reasons behind the name. The third section presents a practical example of some of X-Learning functions. The fourth section shows the X-Learning architectural stack and the reasons why these technologies have been chosen. The fifth section presents the development methodology that has been thought for the project.

* 1. X-Learning overview
  2. Why the name?

The name X-Project has been chosen for three diﬀerent reasons. First of all, X character, in maths, can represent any value. Just as this project’s soul, the X is variable. X-Project can be used to make any kind of web application, from blogs to e-commerce websites. So, this versatility of mathematical X, fits this project soul. Secondly, all experimental elements’ names created by Google during the dawn of Polymer-Project were preceded by a “X-” prefix. Finally, the X-Project is a sort of a tribute to X-Tag: the Mozilla’s web components library [20].

* 1. X-Learning Architecture

A Web application developed by using the x-project toolkit, is a full stack JavaScript Single Page Application.

* + 1. **Server side**

On the server side, an X-Learning app is based on NodeJS (see 2.4) used to create the server environment, MongoDB (see 2.5) used to storage data, and the Web framework Loopback by Strongloop (see 2.6).

Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and eﬃcient. Node.js’ package ecosystem, npm, is the largest ecosystem of open source libraries in the world. NodeJS lets to create a vertical full-stack application in Javascript. The NodeJS asynchronous development scheme increases performances of web applications, by using downtime caused by HTTP requests.

LoopBack generates model API from the models schemas, to let CRUD operations on models. Loopback is the core of the X-Learning server-side. Document oriented API definition guarantees easiness and speed in API creation. Moreover, Loopback, is fully compatible with several DBMS thanks to connectors.

MongoDB is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas. The document-oriented being of MongoDB allows to horizontal scale in really easy way. Moreover, document models, that replace relational database’s rows, are schemaless: this feature gives to MongoDB project high levels of flexibility and manageability.

* + 1. **Client side**

On the client side, an X-Learning app is based on HTML5 Web Components via Polymer-Project by Google. X-Elements lies on the top of this stack. It is a set of Polymer elements for local routing, API request, forms, lists, style and admin panels (as listed further 4.6).

Polymer-Project is one of the most important emerging realities of the moment. Google team has heavily focused his forces on code reusability and on separation between behavior, presentation and content. Code reusability is a direct eﬀect of Web Components structure: the creation of widget that can be completely indipendent facilitates code reuse.

Finally, the union between Polymer and Strongloop has been topped by the creation of the document oriented development process described below 4.5.

* 1. Document-Driven Web Development Process

The process to build a web application based on x-learning toolkit consists of the following four steps: models schemas definition, HTTP RESTful API definition, UI components definition and UI components assembly.

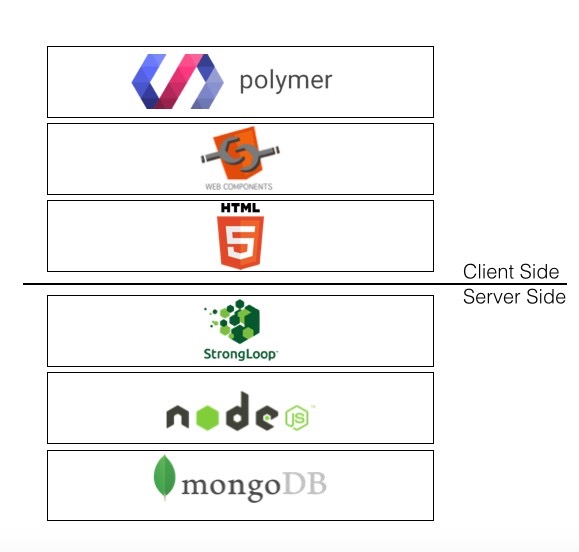


Figure 4.2: X-Project Architectural Stack

* + 1. **1st step - Models schemas definition**

A description of entities, properties, relations and data access policies are defined as JSON documents.

* + 1. **2nd step - HTTP RESTful API definition**

CRUD operations on models are automatically generated by the web framework (on the basis of input JSON documents) and further custom actions can be defined. All of them are exposed as HTTP RESTful API.

* + 1. **3rd step - UI components definition**

Distinct UI components can be defined, or retrieved from a collection of predefined components, configured and adapted. They represent the building blocks of the whole UI.

* + 1. **4th step - UI components assembly**

Distinct UI components are finally mounted to compose the application views. Assembly is kept as simple as possible: it only consists of a composition of HTML5 elements. So, the entire development process results driven by: JSON documents describing entities of the application and HTML template documents describing the UI components.

**Chapter 5**

**Media Management:S3 Component**

This chapter analyzes the component that has been implemented to man- age media in X-Learning via Amazon AWS. The first section present informations about service of Amazon AWS. Between the second and the fifth sections introduces some of AWS service used during the development of X-Learning.

* 1. Amazon AWS

Amazon Web Services (AWS), a collection of remote computing services, also called web services, make up a cloud-computing platform offered by Amazon.com.[2] These services operate from 11 geographical regions across the world. The most central and well-known of these services arguably include Amazon Elastic Compute Cloud and Amazon S3. Amazon markets these products as a service to provide large computing-capacity more quickly and more cheaply than a client company building an actual physical server farm [4].

AWS is located in 11 geographical “regions”: US East (Northern Vir- ginia), where the majority of AWS servers are based,[19] US West (northern California), US West (Oregon), Brazil (São Paulo), Europe (Ireland and Germany), Southeast Asia (Singapore), East Asia (Tokyo and Beijing) and Australia (Sydney). There is also a “GovCloud”, based in the Northwestern United States, provided for U.S. government customers, complementing existing government agencies already using the US East Region.[4] Each Region is wholly contained within a single country and all of its data and services stay within the designated Region.[citation needed]

Each Region has multiple “Availability Zones”, which are distinct data centers providing AWS services. Availability Zones are isolated from each other to prevent outages from spreading between Zones. Several services operate across Availability Zones (e.g., S3, DynamoDB) while others can be configured to replicate across Zones to spread demand and avoid downtime from failures. Amazon web services hold 1.79% market share. As of December 2014, Amazon Web Services operated an estimated 1.4 Million servers across 28 availability zones [18].

* 1. Amazon S3

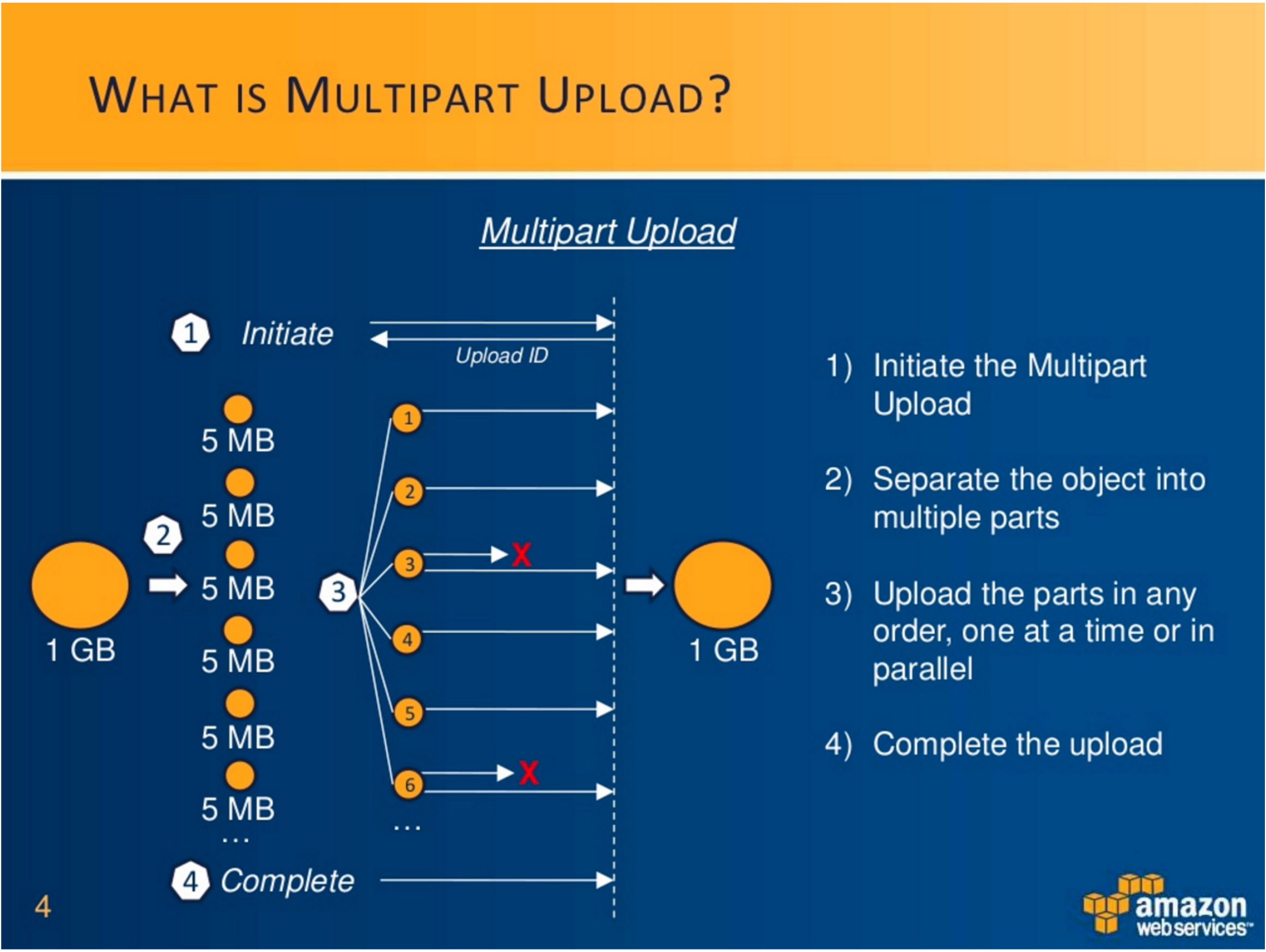
Amazon S3 (Simple Storage Service) is an online file storage web service offered by Amazon Web Services. Amazon S3 provides storage through web services interfaces (REST, SOAP, and BitTorrent).Amazon launched S3, its first publicly available web service, in the United States in March 2006 and in Europe in November 2007 [25].

Amazon S3 is reported to store more than 2 trillion objects as of April 2013.[10] S3 uses include web hosting, image hosting, and storage for backup systems. Amazon S3 provides an API (Application programming interface) for third-party developers. It describes various API operations, related request and response structures, and error codes.[38] Web services interface can be used to store and retrieve any amount of data, at any time, from any- where on the web. It gives any developer access to the same highly scalable, reliable, secure, fast, inexpensive infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers. Today, there are different kinds of file managers for Amazon S3. An effective solution for Amazon provides a user interface to Amazon S3 accounts, files and buckets, allowing to browse, create and delete files and buckets.

**2.1** **Amazon S3 Multipart Upload**

In X-Learning platform the upload video use the Multipart upload API enables it to upload large objects quickly through the parallel upload of the n parts of the object .

Multipart uploading is a three-step process: You initiate the upload, you upload the object parts, and after you have uploaded all the parts, you complete the multipart upload. Upon receiving the complete multipart upload request, Amazon S3 constructs the object from the uploaded parts, and you can then access the object just as you would any other object in your bucket.

You can list of all your in-progress multipart uploads or get a list of the parts that you have uploaded for a specific multipart upload. Each of these operations is explained in this section.

**2.1.1 Multipart Upload Initiation**

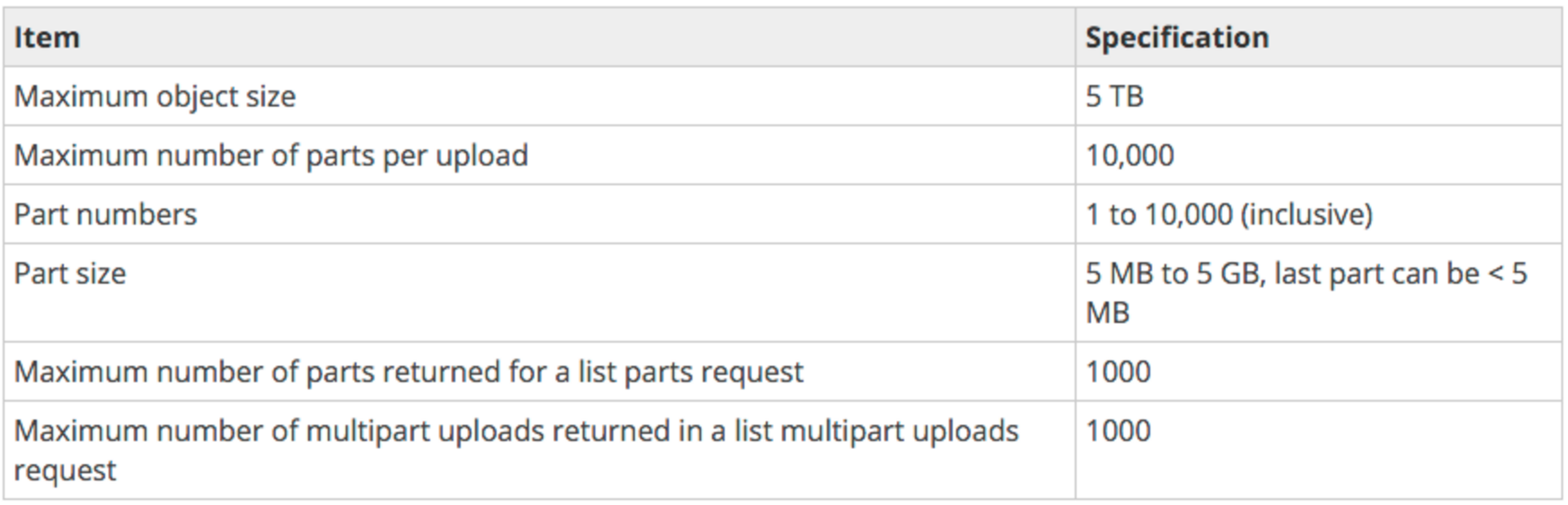
When you send a request to initiate a multipart upload, Amazon S3 returns a response with an upload ID, which is a unique identifier for your multipart upload. You must include this upload ID whenever you upload parts, list the parts, complete an upload, or abort an upload. If you want to provide any metadata describing the object being uploaded, you must provide it in the request to initiate multipart upload.

**2.1.2 Parts Upload**

When uploading a part, in addition to the upload ID, you must specify a part number. You can choose any part number between 1 and 10,000. A part number uniquely identifies a part and its position in the object you are uploading. If you upload a new part using the same part number as a previously uploaded part, the previously uploaded part is overwritten. Whenever you upload a part, Amazon S3 returns an ETag header in its response. For each part upload, you must record the part number and the ETag value. You need to include these values in the subsequent request to complete the multipart upload.

After you upload one or more parts, you must either complete or abort multipart upload in order to stop getting charged for storage of the uploaded parts.

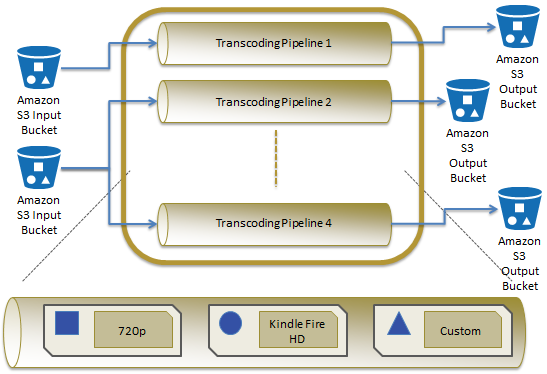
**2.1.3 Multipart Upload Completion (or Abort)**

When you complete a multipart upload, Amazon S3 creates an object by concatenating parts in ascending order based on the part number. If any object metadata was provided in the initiate multipart upload request, Amazon S3 associates that metadata with the object. After a successful complete request, the parts no longer exist. Your complete multipart upload request must include the upload ID and a list of both part numbers and corresponding ETag values. Amazon S3 response includes an ETag that uniquely identifies the combined object data. This ETag will not necessarily be an MD5 hash of the object data. You can optionally abort the multipart upload. After aborting a multipart upload, you cannot upload any part using that upload ID again. All storage that any parts from the aborted multipart upload consumed is then freed. If any part uploads were in-progress, they can still succeed or fail even after you aborted. To free all storage consumed by all parts, you must abort a multipart upload only after all part uploads have completed.

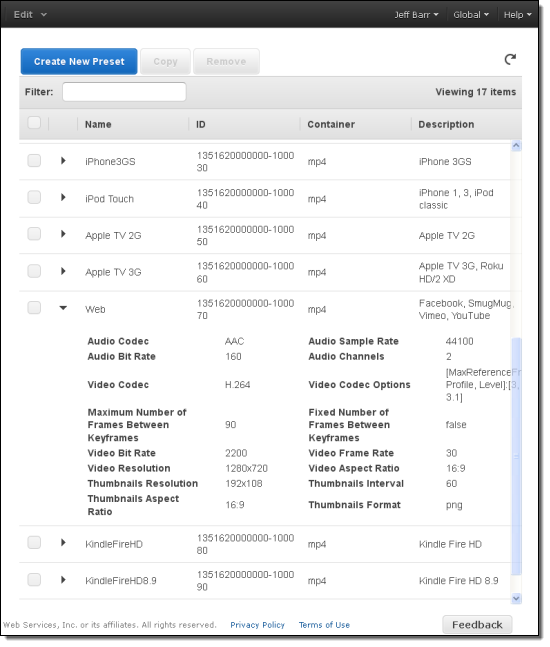
**3. Amazon Elastic Transcoder**

Amazon Elastic Transcoder lets you convert media files that you have stored in Amazon S3 into media files in the formats required by consumer playback devices. For example, you can convert large, high-quality digital media files into formats that users can play back on mobile devices, tablets, web browsers, and connected televisions.

Amazon Elastic Transcoder lets you convert media files that you have stored in Amazon Simple Storage Service (Amazon S3) into media files in the formats required by consumer playback devices. For example, you can convert large, high-quality digital media files into formats that users can play back on mobile devices, tablets, web browsers, and connected televisions.



Elastic Transcoder has four components:

* **Jobs** do the work of transcoding. Each job converts one file into up to 30 formats. For example, if you want to convert a media file into six different formats, you can create files in all six formats by creating a single job. When you create a job, you specify the name of the file that you want to transcode, the names that you want Elastic Transcoder to give to the transcoded files, and several other settings. For each format that you want to transcode into, you also specify a template, known as a preset (see below), that contains the audio and video settings that you want to use for the transcoded file or files.
* **Pipelines** are queues that manage your transcoding jobs. When you create a job, you specify which pipeline you want to add the job to. Elastic Transcoder starts processing the jobs in a pipeline in the order in which you added them. If a pipeline already contains jobs when you create a new job, Elastic Transcoder queues the newest job and begins processing it as soon as resources are available for that pipeline. If the pipeline is already using all of its resources, Elastic Transcoder begins processing the next job in the pipeline when it finishes one of the jobs that it's currently processing. A pipeline can process more than one job simultaneously, and the time required to complete a job varies significantly based on the size of the file you're converting and the job specifications. Accordingly, jobs don't necessarily complete in the order in which you create them.You can temporarily stop processing jobs by pausing the pipeline.
* **Presets** are templates that contain most of the settings for transcoding media files from one format to another. Elastic Transcoder includes some default presets for common formats, for example, several iPod and iPhone versions. You can also create your own presets for formats that aren't included among the default presets. You specify which preset you want to use when you create a job.
* **Notifications** let you optionally configure Elastic Transcoder and Amazon Simple Notification Service to keep you apprised of the status of a job: when Elastic Transcoder starts processing the job, when Elastic Transcoder finishes the job, and whether Elastic Transcoder encounters warning or error conditions during processing. Notifications eliminate the need for polling to determine when a job has finished. You configure notifications when you create a pipeline.

**4. Amazon SNS and SQS**

<http://docs.aws.amazon.com/sns/latest/dg/welcome.html>

Amazon Simple Notification Service (Amazon SNS) is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. In Amazon SNS, there are two types of clients—publishers and subscribers—also referred to as producers and consumers. Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel. Subscribers (i.e., web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (i.e., Amazon SQS, HTTP/S, email, SMS, Lambda) when they are subscribed to the topic.

*In X-Learning platform the Amazon SNS service is use together the Amazon SQS.*

Amazon SQS is a distributed queue system that enables web service applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component. A queue is a temporary repository for messages that are awaiting processing.

Using Amazon SQS, you can decouple the components of an application so they run independently, with Amazon SQS easing message management between components. Any component of a distributed application can store messages in a fail-safe queue. Messages can contain up to 256 KB of text in any format. Any component can later retrieve the messages programmatically using the Amazon SQS API.

Amazon SQS ensures delivery of each message at least once, and supports multiple readers and writers interacting with the same queue. A single queue can be used simultaneously by many distributed application components, with no need for those components to coordinate with each other to share the queue.

One of the resulting tradeoffs is that SQS does not guarantee first in, first out delivery of messages.

*For X-learning platform the order is not important. Indeed when Elastic Transcoder job completed a message was product and send and a this time the subscriber SQS receive the notification and insert message into queue. X-learning server-side read and processed each SQS message to publish the course.*

**5. Amazon Cloudfront**

CloudFront is a web service that speeds up distribution of your static and dynamic web content, for example, .html, .css, .php, and image files, to end users. CloudFront delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so content is delivered with the best possible performance. If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately. If the content is not currently in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

This dramatically reduces the number of networks that your users' requests must pass through, which improves performance. Users get lower latency—the time it takes to load the first byte of the object—and higher data transfer rates. You also get increased reliability and availability because copies of your objects are now held in multiple edge locations around the world.

After some initial setup, CloudFront works invisibly to speed up delivery of your content. This overview includes both the steps you perform before your first user accesses your application or website and how CloudFront serves your content when configuration is complete.

Setting up CloudFront involves a few simple steps:

How You Configure CloudFront to Deliver Your Content

You configure your origin servers, from which CloudFront gets your files for distribution from CloudFront edge locations all over the world.

An origin server stores the original, definitive version of your objects. If you're serving content over HTTP, your origin server is either an Amazon S3 bucket or an HTTP server, such as a web server. Your HTTP server can be running on an Amazon Elastic Compute Cloud (Amazon EC2) instance or on a server that you manage; these servers are also known as custom origins.

If you're distributing media files on demand using the Adobe Media Server RTMP protocol, your origin server is always an Amazon S3 bucket.

You upload your files to your origin servers. Your files, also known as objects, typically include web pages, images, and media files, but can be anything that can be served over HTTP or a supported version of Adobe RTMP, the protocol used by Adobe Flash Media Server.

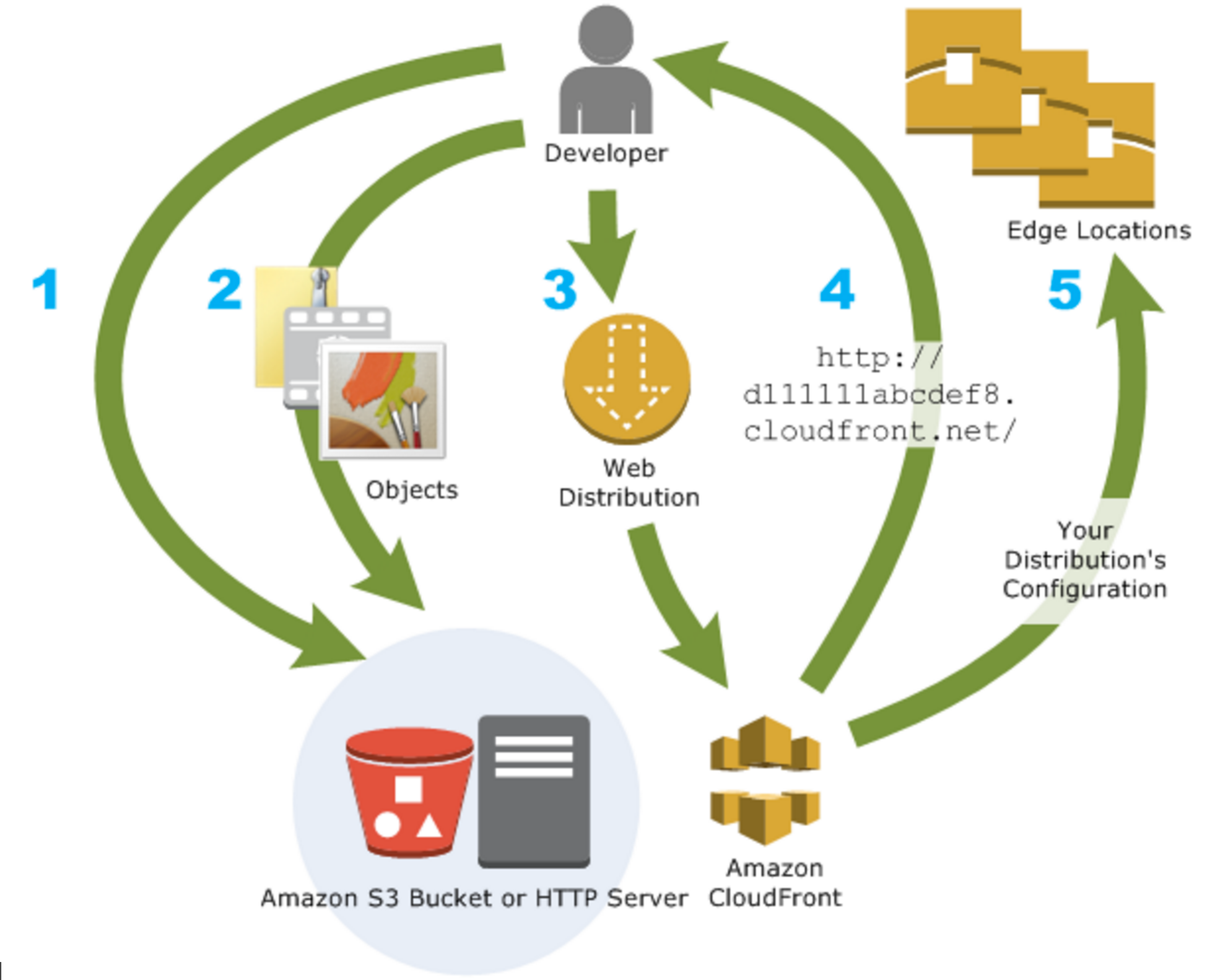
If you're using an Amazon S3 bucket as an origin server, you can make the objects in your bucket publicly readable, so anyone who knows the CloudFront URLs for your objects can access them. You also have the option of keeping objects private and controlling who accesses them.

You create a CloudFront distribution, which tells CloudFront which origin servers to get your files from when users request the files through your web site or application. At the same time, you specify details such as whether you want CloudFront to log all requests and whether you want the distribution to be enabled as soon as it's created.

CloudFront sends your distribution's configuration (but not your content) to all of its edge locations—collections of servers in geographically dispersed data centers where CloudFront caches copies of your objects.

As you develop your website or application, you use the domain name that CloudFront provides for your URLs. For example, if CloudFront returns d111111abcdef8.cloudfront.net as the domain name for your distribution, the URL for logo.jpg in your Amazon S3 bucket (or in the root directory on an HTTP server) will behttp://d111111abcdef8.cloudfront.net/logo.jpg.

You can also configure your CloudFront distribution so you can use your own domain name. In that case, the URL might be http://www.example.com/logo.jpg.

Optionally, you can configure your origin server to add headers to the files; the headers indicate how long you want the files to stay in the cache in CloudFront edge locations. By default, each object stays in an edge location for 24 hours before it expires. The minimum expiration time is 0 seconds; there isn't a maximum expiration time limit. For more information, see Specifying How Long Objects Stay in a CloudFront Edge Cache (Expiration).

How CloudFront Delivers Content to Your Users

Once you configure CloudFront to deliver your content, here's what happens when users request your objects:

A user accesses your website or application and requests one or more objects, such as an image file and an HTML file.

DNS routes the request to the CloudFront edge location that can best serve the user's request, typically the nearest CloudFront edge location in terms of latency, and routes the request to that edge location.

In the edge location, CloudFront checks its cache for the requested files. If the files are in the cache, CloudFront returns them to the user. If the files are not in the cache, it does the following:

CloudFront compares the request with the specifications in your distribution and forwards the request for the files to the applicable origin server for the corresponding file type—for example, to your Amazon S3 bucket for image files and to your HTTP server for the HTML files.

The origin servers send the files back to the CloudFront edge location.

As soon as the first byte arrives from the origin, CloudFront begins to forward the files to the user. CloudFront also adds the files to the cache in the edge location for the next time someone requests those files.

After an object has been in an edge cache for 24 hours or for the duration specified in your file headers, CloudFront does the following:

CloudFront forwards the next request for the object to your origin to determine whether the edge location has the latest version.

If the version in the edge location is the latest, CloudFront delivers it to your user.

If the version in the edge location is not the latest, your origin sends the latest version to CloudFront, and CloudFront delivers the object to your user and stores the latest version in the cache at that edge location.

**Chapter 7**

**Conclusions**

Bibliography

[1] <https://en.wikipedia.org/wiki/Massive_open_online_course>

1. [↑](#footnote-ref-2)