

WCD – Wix Code

A website building system integrating code elements and search engine support

Provisional Patent Application

Contents

Introduction	1
Related patent applications	2
Background	2
System description and architecture	3
Concept details.....	7
Appendixes table	10

Introduction

1. This patent application is directed to website building systems (and other visual editing systems), and describes sophisticated visual editing tools which provide for integration of User Provided Code (UPC) on both the client and the server side, while still generating web sites which may be indexed by search engines and may provide SEO support.
2. This patent application describes multiple concepts discussed in the context of Wix Code, a serverless site development platform being developed as part of the Wix website building system offered by Wix.Com Ltd. The discussion below includes a comprehensive description of such a system. It should be clarified that the Wix Code implementation is not the only embodiment of the concepts and technology, and additional and different embodiments may be constructed in accordance with the disclosure herein.

Related patent applications

3. This application references (or is otherwise related to) the patent applications listed in appendix A1, all of which are assigned to the common assignee of the present application. The disclosure of each of the patent applications and patents listed in appendix A1 is expressly incorporated herein, in its entirety.

Background

4. Applicant has identified a gap in the cloud ecosystem. The landscape of IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service) provides solutions for VMs, Container and Networking, platforms of different types for backend developers, back-ends for mobile developers and ready-made software for individuals and enterprises. What is missing in the middle is a platform for web-sites and web-apps.
5. With the emergence of Serverless (also known as Serverless Computing), there are still no players in this gap. The applicant has developed WixCode, a platform for the development and deployment of web-sites and web-apps, in part, to address this gap. This platform provides elements such as frontend optimized JavaScript, SEO compatibility, WYSIWYG site builder, web methods, and backend JavaScript, as well as request time container boot.
6. In particular, disclosed systems provide the ability to visually create websites while specifying embedded code integrated with the web site for both front-end and back-end code. All of this may be done while preserving the ability of the web site to be indexed by search engines and apply SEO techniques.
7. A website building system may be a standalone system, or may be embedded inside a larger editing system. It may also be on-line (i.e., applications are edited and stored on a server), off-line, or partially on-line (with web sites being edited locally but uploaded to a central server). Disclosed systems may typically be a part of an on-line website building system, though they may be integrated with other types of website building systems.
8. Existing website building platforms based on a visual design environment are typically aimed at users with a skill set such as the one needed to operate Microsoft's Office suite. However, these types of users only build 5% of the sites.
9. Disclosed systems support more complex sites, which typically require a higher skill level to create. Such sites are typically created by designers, script writers, and developers – and include 95% of the sites in the Internet. Appendix A2 includes a presentation further describing this spectrum of site creators and site types.

System description and architecture

10. Appendix A3 includes an in-depth internal technical description of an exemplary system and its elements.
11. Appendix A4 includes an internal technical description of an exemplary system for technical users / developers.
12. Appendix B includes a set of user-facing guides and tutorials describing an exemplary system's specification and use by users / developers.
13. Appendix C includes a set of presentation, tutorials and video scripts providing an overall introduction to an exemplary system.
14. The basic design of an exemplary system (as built in addition to the basic Wix system) consists of 15 sub-systems described below. A 16th "sub-system" is the User-Provided Code (UPC) which may be provided by the site designer and executed as part of the system (with the appropriate encapsulation and separation in place).
15. These exemplary sub-systems may have client-side presence, server-side presence, or both. The exemplary sub-systems may include, among others:

#	Sub-system name	Acronym	Client	Server
1	Compute grid	CGD		V
2	Elementory	ELM		V
3	Wix Data	WDT	V	V
4	Web methods / RPC	WBM	V	V
5	Zero Setup IDE	ZSI	VVV	V (file system)
6	Editor platform	EPT	V	
7	Client side Web worker	CSWW	VVV	V
8	Router	RTR		V
9	Dynamic pages	DYP	V	V
10	Data bindings	DBD	V	
11	Content manager	CMGR	V	
12	Dynamic SEO	DSEO		V
13	Compute functions	CFN		V
14	Wix DB	WDB		V
15	\$W API	\$WAPI	V	V
---	User Provided Code	UPC	V	V (for SEO)

16. These exemplary sub-systems, and their example functionality, can be briefly described as follows:

#	Sub-system name	Acronym	Description
---	-----------------	---------	-------------

1	Compute grid	CGD	Back end server which runs the dockers ⁽¹⁾: • Does not include the part inside dockers; • Does docker recycling & fast docker booting; • May provide specific services for Elementory
2	Elementory	ELM	The application server inside each docker: • node.js based; • Used to prepare client-side scripts – now this was moved outside as a separate service in compute grid
3	Wix data	WDT	The DB underlying the system, including: • Client & server API + implementation; • Permission system; • Hook system (customizable callbacks) • DB drivers – connect to underlying DB's (such as Wix DB below or alternative DB's); Regarding schema use: • Uses "soft" schema by itself – the schema is provided from above. • In the API's there is no schema; in the application level there is; • The content manager may have a cell which diverges from the schema; thus we have use cases of "handle non-compliant data"; Provides two DB interfaces – sandbox & live; Implemented with a single DB + differences for the 2nd one; preserve O(1) operation; See comments regarding algorithms below.
4	Web methods / RPC	WBM	RPC from client to server: • Automatic wrapper for Ajax; • Provides security services; • Error & log messages from server are automatically sent to the client – to the browser console (for debugging); Thus provides a "single system image" feel when debugging;
5	Zero Setup IDE	ZSI	Zero-setup IDE: • On-line IDE integrated with editor; • Supports code completion; • Specific support for media (opening media manager for images / video / docs URL's etc.).
6	Editor	EPT	A new way to add application to Wix site:

¹ <https://www.docker.com/what-docker>
<https://www.docker.com/what-container>

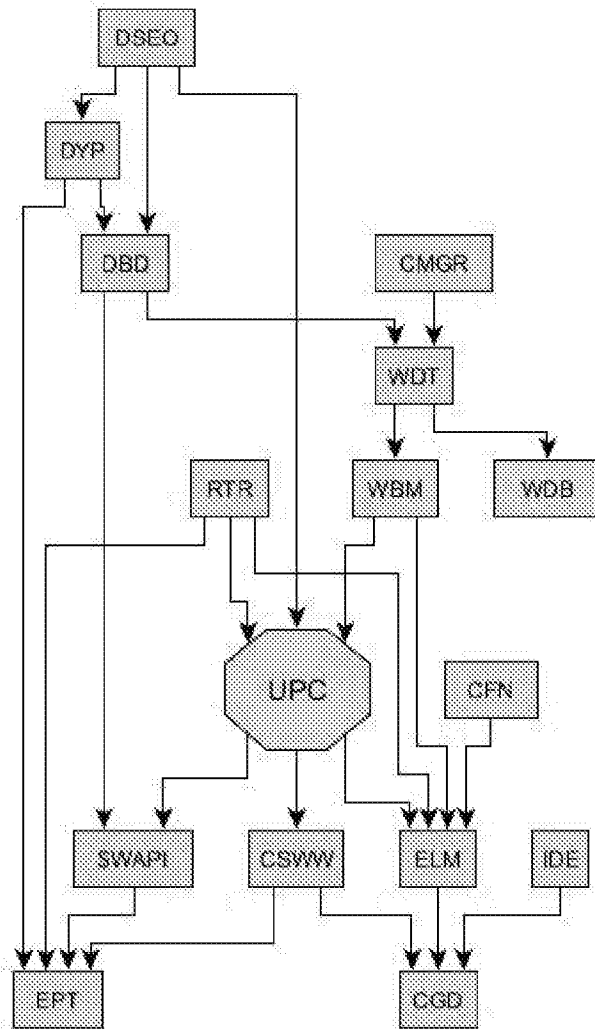
	platform		- Based on web worker²; - Web worker in an iframe; runs the app code; - Controller – connects to Wix components; - Includes interface to API's; - Has router (infrastructure); Code backend; can control URL's & selection of pages;
7	Client side web worker	CSWW	RT code inside the web-worker separate thread on the client which runs the client user code: - Consists of a series of API which allow code in the web worker iframe which operates and modifies the web site; - May be limited to changing data in component and changing visibility; - Also allows control of some types of components, e.g. for a gallery change to show a specific item (next item, item X, ...). - May be extended to include modifying the site itself, e.g. add/remove components, change component attributes;
8	Router	RTR	Router infrastructure; allows user to write router code directly;
9	Dynamic pages	DYP	Application with its own router; The router: given a collection of Wix Data, creates items and category pages; exposes the data through data bindings; These are actually newly created pages with their own URL's – this is the situation for all routers; Can be extended to other page types;
10	Data bindings	DBD	Application which defines its own controller; its controller manages is a dataset and connects data in the database with UI component;
11	Content manager	CMGR	Excel like editing tool for Wix Data. - Supports soft schema. - Thus, the system may have a string value inside a number cell; The cell would be displayed using a string displayer, but marked with a red line (similar to speller); on hover, display an attached pop-up with the string and let user fix it;
12	Dynamic SEO	DSEO	Complete SEO solution for Wix Code; - Applies SEO to Dynamic pages, data bindings & API's; - Run the worker above (usually running in the client) in the server environment instead;

² https://en.wikipedia.org/wiki/Web_worker

			• Provide processed search-engine friendly pages;
13	Compute functions	CFN	The ability of the back-end to expose functions directly to the front-end; call a function and get a response; a-la HTTP backend;
14	Wix DB / DocStore	WDB	Scaled out Mongo DB underlying the system; Supports externally managed sharding ³ of data;
15	\$W API	\$WAPI	API used by the user to affect the site's visuals: • The main channel used by DBD to affect component attributes, visibility etc.; • Similar to jQuery.
---	UPC	UPC	User provided code: • This is the actual code written or supplied by the user as part of the web site; • Runs inside the web worker – typically on the client but may run on the server for SEO;

17. The relationship between the various sub-systems can be visualized as follows:

³ [https://en.wikipedia.org/wiki/Shard_\(database_architecture\)](https://en.wikipedia.org/wiki/Shard_(database_architecture))



18. A detailed description of each of these exemplary sub-systems is included in appendix A5.
19. Appendix A6 includes a specific “database overlay” algorithm for the implementation of one embodiment of WDT above. An alternative embodiment may use “tombstone records” (signifying records which have been deleted) in the discussed “Plus DB” instead of using a “Minus DB”.

Concept details

20. Applicant has identified a number of areas, functions, and capabilities described in greater detail herein. These include the following:
 - 20.1. Custom Back-end Functionality in an Online Website Building Environment.
 - 20.2. Dynamic Preview of Database-Populated Web Pages.
 - 20.3. Editing a Database during Preview of a Virtual Web page.

- 20.4. On-Demand Web-Server Execution Instance for Website Hosting.
 - 20.5. Common Database for Live Operation and Testing of a Website.
 - 20.6. Automatic Layout Recommendation Engine (Magic Layout).
 - 20.7. Automatic Website Design Recommendations to Improve SEO.
 - 20.8. Segregated User-Uploaded Plugins Avoid Infection Across Co-Hosted Websites.
 - 20.9. Segregating User-Uploaded Plugins for Websites.
 - 20.10. User-Supplied Plugins Extend Online Editor Functionality.
 - 20.11. iFrame Editor Interface Segregated from Design Tools.
21. Appendix D provides further details and disclosure regarding each of these concepts. Below are some specific additional clarifications regarding some of the concepts.
22. Custom Back-end Functionality in an Online Website Building Environment:
- 22.1. The system may provide specific search engine spider support through a script or marker included in the HEAD tag of the generated pages.
 - 22.2. Such a marker or script may direct the spider to use our specific dedicated web service, helping the make the pages indexable.
23. On-Demand Web-Server Execution Instance for Website Hosting:
- 23.1. This concept involves rapid boot of docker containers running a web server instance configured for a specific site (including the specific plug-ins and server-side user provided code for this site), and doing so quickly enough to respond to an HTTP request without timeout, and in a total timeframe acceptable to users.
 - 23.2. Starting a virtual machine typically takes a number of minutes. Starting a Docker container still takes 2-3 seconds. Both of these times are not acceptable.
 - 23.3. In disclosed systems, a pool of standby dockers (e.g., containers) may be provided containing all relevant non-site-specific code but without any user/site specific code.
 - 23.4. The system may listen on a live port for a request to attach to a server for the specific web site. If there is a live docker associated with the specific web site the system may attach to it. Otherwise, it can use a standby docker from the pool and inject the requested content into it, or instruct it to load such content.
 - 23.5. The system may drop standby containers if a site has not been active for a given period (such as 15 minutes).
24. Common Database For Live Operation and Testing of a Website:
- 24.1. This concept involves having staging & live data running on the same DB.
 - 24.2. As noted above, an example algorithm is detailed in appendix A6.
 - 24.3. Any writing to the DB by the development system may create a copy of the data. This copy may be the one visible to the development system. The development data may not be visible for the live site.
25. Segregating User-Uploaded Plugins for Websites:

- 25.1. The concept discusses the use of iFrames to take content from one site and incorporate into another site securely. However, such content incorporation may be implemented using a number of methods, including:
- 25.1.1. Transferring the content via AJAX.
 - 25.1.2. Direct inclusion of the remote content subject to static code analysis, e.g. analyze the JS code to see if it is safe.
- 25.2. Disclosed systems use a smaller (and typically invisible) iframe to run the plugin code, while the plugin UI itself may be native to the website building system. This is different from the arrangement used with TPA's (Third Party Applications), in which some or all of the TPA's UI is implemented in an iFrame, though the TPA may still communicate with other native components in the page hosting the TPA (as discussed in the TACA Wix patent application – see appendix A1).
- 25.3. Plugins also differ from TPA in that plugins typically involve only user provided code (which runs on the client side only). TPA code on the other hand runs on the TPA provider server.
- 25.4. Thus, disclosed systems may provide the following benefits:
- 25.4.1. Allow installing a plugin into a site so it adds functionality but doesn't jeopardize the web site itself due to code isolation.
 - 25.4.2. The plugin can't access the cookies & DOM of the site – and this security measure can't be bypassed like the (limited) encapsulation provided by Web Components⁴ or similar systems.
 - 25.4.3. Disclosed systems allow installing multiple plugins so they won't interfere with each other (e.g. by placing them in separate iFrames).
 - 25.4.4. Disclosed systems may allow installing a plugin which extends another plugin – using a single iFrame for both, or using an API between multiple iFrames. Such plugins are not isolated or sandboxed from each other, e.g. they may be part of a plugin family provided by the same vendor.
 - 25.4.5. Disclosed systems may provide a secure plug-in sub-region with a rich API which is decoupled from the site itself, allowing multiple plugins to reside in the same area, and supporting a rich API through the \$w API described above (possibly using a controlled cross-iFrame API as described in the TACA patent application – see appendix A1).
- 25.5. This arrangement differs from regular iFrames, as iFrames do not have a developed API, and do not interact with the hosting WBS (website building system) or WBS page. It is also effective in preventing CSRF (Cross-site request forgery), cookie theft and XSS (Cross-site scripting). However, it can only be used in an on-line WBS.
- 25.6. This should be contrasted with existing prior art systems, which typically rely on having correctly written plug-in, or are limited to specific plug-ins.

⁴ <https://www.webcomponents.org/>

- 25.7. The implementation may rely, for example, on hooking on the gateway for PostMsg and filter the specific messages. Thus, disclosed systems may also implement a user-oriented permission technique, which allows the user to precisely control the access permissions granted to any plugin. This is similar to the way in which a Smartphone operating system provides precise control over the permission granted to any installed applications (e.g. "Can App XYZ use the GPS / access your contacts?"). However, this is done without requiring downloading of a copy of the plugin before installation.
- 25.8. It should further be noted that disclosed systems, as an on-line WBS, may store plugins in a single place, with all installations of the plugin updated immediately when the plugin is updated. This is in contrast with existing off-line prior-art systems in which each plugin installation is separate and should be updated by itself.

Appendixes table

Appendix	Description
A1	List of related patent applications
A2	The spectrum of web site creators skill levels
A3	In-depth internal technical description of the system
A4	Internal technical description of the system for technical users / developers
A5	Description of the various sub-systems as implemented in Wix Code.
A6	A "database overlay" algorithm for WDT
B	A set of user-facing guides and tutorials describing the system's specification and use by users / developers
C	A set of presentation, tutorials and video scripts providing an overall introduction to the system.
D	Concepts appendix