Tampere University

Krishna Bagale

Web Application Development with ValOS Stack

Faculty of Information Technology and Communication Sciences
M. Sc. thesis
December 2020

ABSTRACT

Krishna Bagale: Web Application Development with ValOS stack
M. Sc. thesis
Tampere University
Master's degree program in software development
December 2020

Web applications have become an essential component of everyday life in today's digital business world. By utilizing web applications, businesses can develop and become simpler and achieve their objectives in a speck of time. These applications can help target numerous clients and customers at a time and interact with them in real-time. ValOS is an open-source technology stack allowing scalable and secure web and IoT applications to be built in a fraction of time with only basic skills in web development. It is relatively easy to create and deploy full-fledged web applications with ValOS and integrate the development with other frameworks or libraries such as Bootstrap and React. ValOS is written in JavaScript and ValOS is compatible with the JavaScript language. ValOS is an extension or a dialect of JavaScript.

The world's first website was published in 1990. At the time the available website was not graphical, but text-only and included some links. In 1992 the first image was published online, which was edited with photoshop. Cascading Style Sheet (CSS) was introduced between 1996 and 1999. Before the introduction of the CSS, the styling of websites was done with inline styling. CSS was the language to differentiate the content and presentation. Later several frameworks and libraries were involved in making the website visually presentable to the public. State handling is the number one source of complexity in large-scale software systems. Event sourcing pattern, a ValOS concept simplifies state management in complex domains. Instead of storing the current state of data, it keeps a record of events that describe changes to that data.

Frameworks allow beginners to make progress and experts to progress quickly. The issues related to using the frameworks are DNS issues and network connectivity, slow servers and loading time, lack of load balancing, poorly written code, failing to optimize bandwidth usage, traffic spikes, specific HTML title tags, etc. Among all the modern-day web frameworks and libraries, the ReactJS library stands on top. ReactJS is a JavaScript-based library developed by Facebook, Inc. Django, RoR is the most used back-end framework. The purpose of this thesis is to introduce Valaa technology, its development platform, and the new dimension that Valaa brings to web development. The example of a networking web application is used to introduce the Valaa platform in depth.

Keywords: Valaa, ValOS, JavaScript, ReactJS, Django, RoR, Framework.

ACKNOWLEDGMENT

This master's thesis was written at Tampere University as part of the curriculum of the degree program in the Faculty of Information Technology and Communication Sciences. This project was made under the supervision of Timo Poranen and Ville Ilkkala from Valaa technologies, who guided and supported the author a lot by providing helpful advice as well as resources. There has not been any research on Valaa's platform or the implementation of a web application in a thesis. Therefore, the idea of writing a thesis on the subject matter was welcomed by the company, mentor, and managing director Ville Ilkala and supervisor Timo Poranen. I am forever grateful and indebted to them. The thesis seminars conducted by the head of the department Zheying Zhang helped in outlining the research plans, research questions, and finding out the necessary literature articles.

Tampere, 01.12.2020
Krishna Bagale

TABLE OF CONTENT

LIST OF ABBREVIATIONS

AngularJS	JavaScript front-end web application framework.
Apache	Cross-platform web server software.
App	Application.
API	Application Programming Interface.
AR	Action Research.
AWS	Amazon web server.
cURL	"Client URL" named command-line tool to transfer data.
CA	Certificate Authority.
Chrome	Google browser.
CRUD	Create, Read, Update, Delete operations.
CSS	Cascading style sheets.
DOM	Document Object Model.
Django	Python framework.
DS	Design science.
DSR	Design science research.
DynamoDB	Amazon database service.
Firefox	Web browser developed by the Mozilla Foundation.
FTP	File transfer protocol.
GIF	Graphics Interchange Format.
GraphQL	Query and manipulation language for APIs.
HTML	Hypertext Mark-up Language.
HTTP	Hypertext Transfer Protocol.
IDE	Integrated Development Environment.
IE	Internet Explorer browser.
IndexedDB	JavaScript API in web browsers for managing a NoSQL DB of JSON objects.
IoT	Internet of Things.
JS	JavaScript programming language.
JSON	JavaScript Object Notation.
JSX	Syntax extension to JavaScript.
LAMP	Linux, Apache, MySql, Php.
MEAN	Mongo, Express, AngularJS, NodeJS web stack.
MeteorJS	Web framework.
MERN	Mongo, Express, React, NodeJS Web stack.
MEVN	Mongo, Express, VueJS, NodeJS Web stack.
MQTT	Message Queuing Telemetry Transport.
MVC	Mode-, View, Controller software design pattern.

NGINX	Web server software for web serving, reverse proxying, caching, and more.
NodeJS	JavaScript runtime environment.
NoSql	Not only SQL.
Npm	Node package manager.
OS	Operating System.
PDF	Portable Document Format.
PNG	Portable Network Graphics.
RDBMS	Relational Database Management System.
Ruby	Programming language.
SEO	Search engine optimization.
SPA	Single Page Application.
SSL	Secure Sockets Layer.
SQL	Structured Query Language.
TLS	A type of digital certificate issued by a Certificate Authority (CA).
TCP	Transmission Control Protocol.
UI/UX	User Interface/User Experience.
URL	Unified Resource Locator.
URM	Unified Resource Modeling.
VS	ValoScript, Js -like syntax for ValOS.
VSX	Xml/Html -like syntax for ValOS.
Valaa	Company’s name.
VALK	Intermediate query language.
ValOS	Valaa’s Stack Architecture.
XHTML	Extensible hypertext markup language.
XML	Extensible Markup Language.
WordPress	CMS system is written in PHP and paired with MySQL.
W3C	World Wide Web Consortium.
Zero	Valaa’s online IDE.

1 INTRODUCTION

The adaptation of cloud computing solutions is gaining momentum, and large data calculations are done on servers instead of the users' devices. Web application development is becoming popular because users are shifting from personal computers to mobile devices and from native applications to web browser-based alternatives. This study focuses on the development of a web-based application using the ValOS stack and compares it with the trending JavaScript web framework MeteorJS.

Several web frameworks and libraries are constantly evolving to supply webpages with efficiency, agility, compatibility, responsiveness, seamlessness, inclusiveness, lightweight, and adaptability. Developers are allocated with multiple choices of frameworks, which can easily be confusing. Selecting favorable and convenient frameworks or libraries is utterly complicated and entangles developers in the frequent changes in new technologies.

The networking web application project is an idea that a client of Valaa Technologies wanted to have for their community and was welcomed by the company and myself. This thesis introduces the ValOS in brief and then goes into the implementation process in greater detail. The ValoScript language is similar to JavaScript so a fundamental understanding of the JavaScript language and a minor experience with full-stack web development using either MEAN, MERN, or MEVN stack is recommended.

Recent web engineering research has focused on the undertaking of developing web applications, while many assumptions of what a web application is—that is, the variety of manifestations a web application may take—have been left unchallenged to a moderately high degree [Iskandar et al., 2020]. No research, to the developer's knowledge, has focused on attempting to document to what variety and extent of web application architectures exist, thereby limiting the boundaries of practices in finding out the core scheme of web applications.

Web application architecture has changed, and many frameworks are introduced for 10 years. Nonetheless, many of these frameworks take an approach that may be described as prematurely prescriptive, based on a well-intentioned desire to provide actionable information and ease of access, but with insignificant amounts of real-world data to support the theoretical aspects. Web application frameworks are built with the philosophy of providing the ultimate perspective on a certain limited web application quality questioning the security issues. An alternative approach would see frameworks as complimentary, presenting particular viewpoints that strengthen or inform each other, and therefore provide a more nuanced view to the user of those frameworks, leaving decisions to the user.

Many web applications rely on storage systems as well as the client-server model of communication that may contain the complexity to establish the connection and could be hard to manage the real-time features. ValOS takes a different approach of streaming events between the web resources, and the users do not need to worry about managing the database system and thus they can focus on building the core implementation of the application and save a lot of

time. Web applications are essentially distributed systems, as they rely on the client-server model of communication [Iskandar et al., 2020].

State handling is the number one source of complexity in large-scale software systems. Web applications are often constructed in such a way that they contain no state, and can therefore be trivially scaled, pushing the real concerns of the distributed systems field to the data layer, which relies on commodity systems, leaving the properties of the web application as a whole uninvestigated. An event sourcing pattern simplifies state management in complex domains. In ValOS, instead of storing the current state of data, event sourcing keeps a record of events that describe changes to that data [Kiiskinen, 2019].

The motivation for this research is due to the interest in the ValOS stack, which aims to attract the author with its solutions in web development as it is well equipped with the emerging tech concepts. This research provides consistent data sufficient to be able to formulate a clear picture of web application frameworks and stacks, as well as a starting point for using web application stacks. The objectives of this research are to emphasize the benefits and limitations of web frameworks and stacks by introducing the ValOS stack and comparing it with MeteorJS (JavaScript framework). Thereby, software developers can understand if it is in their benefit to use a framework in their project. This research also aims to help business organizations and educational institutions to find out if understanding the web development concepts that Valaa Technologies is bringing forward could contribute to their students and teachers.

The first and second chapters outline the thesis introduction, research questions and methodology respectively. The third chapter demonstrates the important literature and theory on web application development in general as well as frameworks and software quality. The theory is needed to support the case project and arguments. The fourth chapter introduces the ValOS system. The fifth chapter demonstrates the networking web application development with ValOS stack and compares the ValOS stack with MeteorJS framework. Chapter five will tie together the evaluation of the existing literature. Finally, the final chapter will conclude the thesis, summarizes the outcomes in general, and also discusses further development.

2 RESEARCH QUESTIONS AND METHODS

This research focuses on the general history and implementation of normal web applications with the modern-day web frameworks, general internet security and web crawlers, libraries, and stack, and how they differ from what Valaa is offering. The benefits of using the ValOS stack, its challenges, performance, and opportunities, the use of a matchmaking algorithm, use of plain JavaScript, HTML, CSS with ValOS will be clarified. The clarity of the comparison between modern web stacks such as MEAN (Mongo, Express, AngulasJS, NodeJS), MERN (replacing AngularJS by ReactJS) with ValOS is understood well after the completion of the same networking web application project using MERN and ValOS. The ValoScript language is similar to JavaScript, so a fundamental understanding of the JavaScript language and a minor experience with full-stack web development using either MEAN, MERN, or MEVN stack is recommended. Given the above information, the research questions of this study are:

- What are the reasons behind the use of web frameworks?
- How to develop a web application using ValOS?
- What are the differences between ValOS and MeteorJS?

This thesis introduces the ValOS stack and MeteorJS framework to illustrate the uniqueness of two different technologies in modern-day web application development. The study design is a process, which is carried out while building the application. All the ideologies of efficiency, security, agility, compatibility, seamlessness, lightweight, and adaptability of the application will be analyzed. ValOS has multiple layers of current and upcoming security features. Cryptographic content security, including end-to-end encryption, is in. General data security is handled using a permission system.

The design science research methodology will be used throughout the research. Design science research is conducted to solve problems identified in the organization [Hevner et al., 2004]. It is based on a rigorous design process to solve the problem observed, to contribute to research, to evaluate designs, and to communicate the results to the relevant public [Hevner et al., 2004]. Artifacts include any objects designed to identify research issues, such as constructions, models, methods, instantiations or social innovations, or new features of technical, social, or information resources [Offermann et al., 2009]. “In the design-science paradigm, knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact.” [Hevner et al., 2004].

Table1. Design Science Research Guideline [Offermann et al., 2009; Van Aken et al., 2005].

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1 lists the DS research guidelines and explains the listed guidelines. In this thesis, the networking web application will be developed under the objectives defined in DS research. During the software development of the application, the guidelines mentioned in Table 1 will be followed. According to Van Aken, the principal aim of design science research is to grow skills that the experts of the discipline can use to design solutions for the challenges of their field [Tripp, 2005].

Hevner states that “the main purpose of design science research is achieving knowledge and understanding of a problem domain by building and application of a designed artifact” [Iivari et al., 2009]. In DS, the idea is not to accumulate theoretical knowledge but to discover and solve the problem [Holmström et al., 2009]. The DS research methodology goes by the motto of “build and evaluate”. Understanding the in-depth advantage of implementing a framework in web application development requires a comprehensive study on all the influencing factors. To acquire the result, this thesis uses action research as its main research methodology besides the design science research methodology.

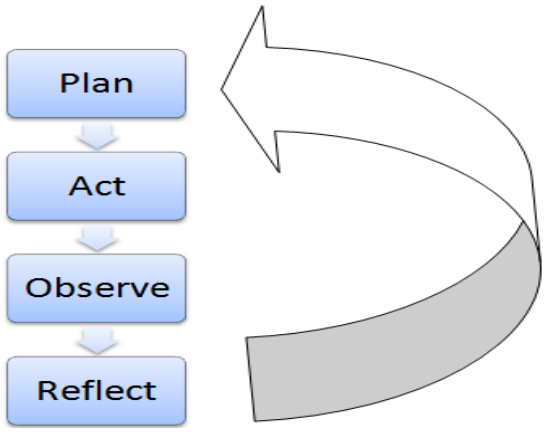

Figure 1. Action Research Methodology [Ilvari et al., 2009].

Action research is an iterative process of obtaining the objectives by the motto of "learning by doing", which is a relatively effective method for constantly planning, acting, observing, and reflecting the outcomes. Figure 1 illustrates the process that is used in action research methodology. With the constant change in contemporary and modern-day technology, it is wise to evolve with frequent commits and observation and repeat the process as long as the desired goals are not met. Originally founded by Kurt Lewin, action research is known simply as "learning by doing". Researchers try to approach the problem and attempt to resolve it with constant commits. Their effort is collected and analyzed during the process. The difference between the DS and AR methodologies is that the DS follows the practice of inventing something new, implementing it, and evaluating the results, in comparison, the AR approach follows the practice of causing an intervention and recording what happened iteratively. Both the methodologies are vitally important in successfully carrying out the implemented project in this thesis.

To cope up with the expanding speed of internet technology and users' needs, web-based applications need to become more and more adjustable and able to perform various actions. The application would get outdated easily without a frequent update and the ability to scale up to meet up with demands. The scope of the thesis is limited to web application frameworks and the project carried out with a ValOS stack. Although some other frameworks and libraries such as MeteorJS and ReactJS library are studied and implemented separately to compare the differences between them, these studies only give the general overview but do not dig deep into the development process itself.

In this thesis, using the ValOS stack in a networking web application project will be compared against the JavaScript framework MeteorJS to introduce and point out the advantages and disadvantages of the stack. The results and the findings are presented to the relevant parties to show the concept and the architecture of the ValOS stack. The application development process with MeteorJS will not be discussed and will be performed separately for learning and finding the comparison topics.

3 WEB APPLICATION

Mosaic 1.0 was the world's first graphical website introduced in 1993 with the landing page. In 1994 one of the world's most popular browsers Netscape was published, offering easy access to the public by introducing animated Graphics Interchange Format (GIF) images and tables. To create a common standard for web developers, W3C was established. Since then CSS has become a foundation of today's web design and development. A web application is a piece of software that can be accessed from a browser. A browser is an application that is used to browse the internet. Firefox, Google Chrome and IE are few example browsers.

A web server can host multiple web applications. A web server processes incoming network requests over HTTP and several other related protocols [Madasu et al., 2015]. A browser is commonly called a Web client or a User-agent. Web clients are not only browsers, but any application such as cURL or Telnet which can communicate with a web server is also a client. For a client and server to communicate with each other, both parties need to use the same set of defined rules which is known as a protocol. HTTP, FTP, WebSocket are the standard protocols widely used these days, and each of the protocols is suitable for different kinds of tasks. For example, the FTP protocol is used for transferring files over the internet. The HTTP protocol is a highly used protocol among all the protocols available [Kulesza et al., 2020].

Most of the time the clients and the web servers communicate with each other using the HTTP protocol. The web server does not keep track of the client's request that it has received via HTTP or the document that the webserver sends back to the web client as a response, so the HTTP protocol is known as a stateless protocol. Every time a client sends a request to the webserver or reloads the web page, the webserver receives the request as a brand-new request. A web client can access the web resource documents through the webserver.

A web resource can be static or dynamic. A resource that does not change is static, for example, a static file that is in the web server or a hard disk to which a web server has got access. Upon receiving the request by the client, the web server could pass the static file as a response. The dynamic resource is generated promptly by the web server when a request for a dynamic web resource comes from a client. A dynamic resource is widely used for real-time features when there is a need for embedding real-time data to be fetched and displayed. For example, weather applications and real-time transportation applications are dynamic as the data is retrieved from the API. In this URL https://www.bagalekrishna.com/tie19/93/web-app-general-view.hml, HTTPS is a protocol, www.bagalekrishna.com is the webserver identifier domain name, tie19/92/web-app-general-view.html is the path of the resource in the webserver.

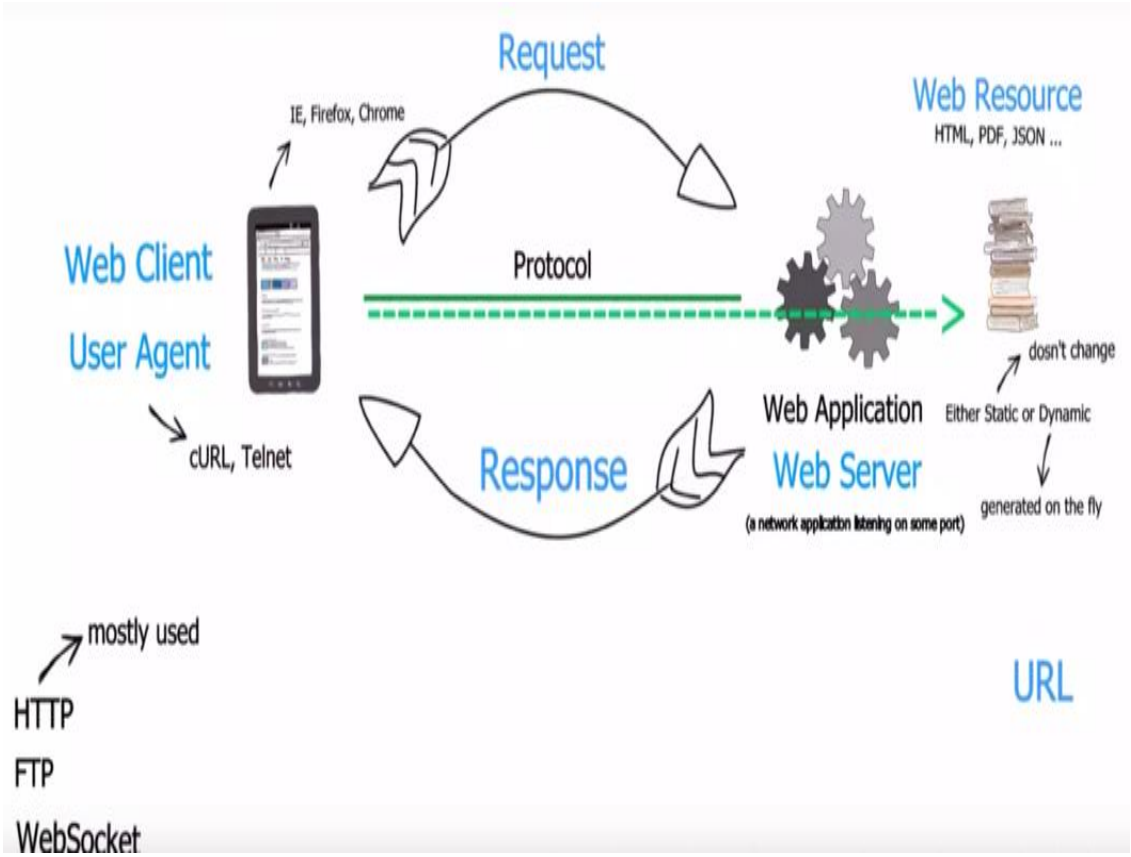

Figure 2. The architecture of the Network and Client-Server model [Zhang Hai 2013].

A web resource commonly referred to as a resource, is a document that is hosted by a web server. Figure 2 demonstrates the architecture of the network and client-server communication model, the client request and server response flow as well as the web resources. A web resource can be a HTML, PDF, JSON or an XML file document or any kind of document, which is hosted by a web server. When a client tries to load the URL, the client requests to establish a communication channel with the server and the server replies by giving back a HTTPS, TLS or SSL certificate.

The certificate includes a list of information, for example a digital signature, domain name and the owner of the domain. After the client is presented with the certificate from the server, there comes a question as to how the client knows if the server is correct or not. The digital signature is being used to verify the public key of the server, the domain name, and the owner so that the client-server connection can be opened through the URL.

The third-party certificate authority (CA), which has nothing to do with the connection between the client and the server, signs the certificate that the webserver holds [Stephen, 2014]. The client before even contacting the server trusts the CA. The client in its operating system or the browser holds a list of CA. The client recognizes all the certificates that are signed by the CA. The computer that we are working on contains in advance the CA information and holds several certificate authorities. So, the client depends on CA, the server also depends on CA and

because of that, the client and the server extend their dependence to trust on one another and the connection gets established.

The bigger question arises as to how to trust the CA itself. The certificates are created only if the server asks for them to create. The CA makes sure that the server is asking for the certificate by getting the necessary information from the owner of the site. So, if anyone goes with the administrative proof of the website to create a certificate with the CA, the CA can then provide the certificate. The email address is used to transfer the information between the server owner and the CA, so whoever has the email access can go to the CA and get an access certificate and because of that, the system is utterly fragile.

Hypertext Transfer Protocol Secure (HTTPS) means the connection over a computer network is secure. The connection being secure itself means that the server and the client where the site is loading have established a connection, in which no one but the connecting parties can see the flow of information between them because the communication protocol is encrypted using Transport Layer Security.

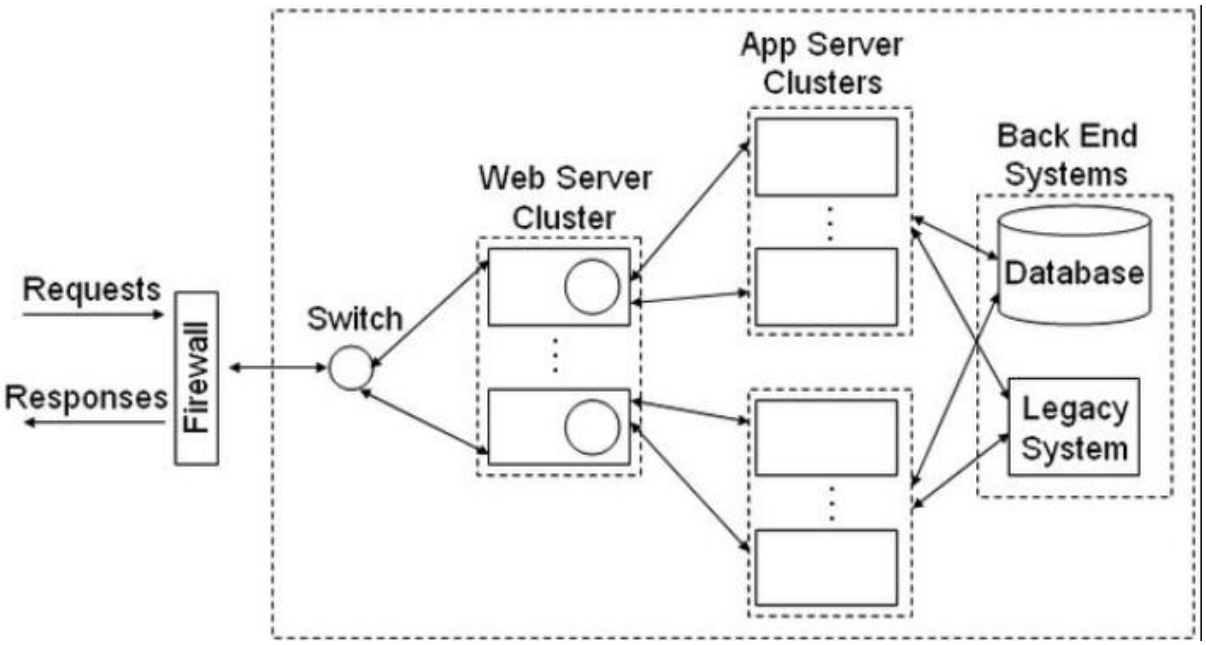

Figure 3. Web application architecture diagram [Dutta et al., 2007].

Figure 3 shows the overview of a web application architecture. Whenever the user visits a web application, the first thing that is shown in the view is front-end. The responsibility of the front-end is to collect the data from the user such as a button click and send information to the backend. The back-end acts upon the given data from the front-end in order to process, store and return the processed data to the user [Shklar et al., 2009]. The web server is an agent, which acts upon the data sent by the user. A web server comes with the plugin. The plugin helps to

run the code in various languages. A web server has a responsibility of serving the files over the Internet to the user.

A web server can directly communicate with the file system. A web server upon getting the request to serve the client with the static file, it can directly retrieve the data and display to the user. In that case the web server does not need to get the dynamic data. If the user requests the data that has to be processed and retrieved from the database, the web server communicates with the database and responses the data. The loading time for a web server to retrieve the dynamic data is higher as compared to the static data [Iyengar et al., 2000]. Web servers also come with several functionalities, for example a feature to plug in the application logic. The web server can connect to store the data into the database by connecting with Object Relational Mapping (ORM) or Connector. Once the Application Logic finishes performing its task that Business Logic requires, it returns the data to the web server.

A web server can process several requests concurrently [Edge Jr. et al., 2010]. In order to enhance the performance and fulfil the workload demands, modern web servers invariably perform multiple tasks at once. The modern web servers create a poll for incoming requests and depending on the time it takes for each request; it asynchronously performs the tasks. In most of the modern web servers, the capacity to hold the incoming requests is high to ensure the acceptance of a request. The changes on files that are on server requires the restart of a web server. In order to display the same dynamic data upon reloading the web browser, a web server caches the dynamic content.

It is relatively easier to cache the static content than event-driven and dynamic content. The static content are simple HTML pages, images, CSS and JavaScript files. Event driven content consists of blog posts and news articles that are not saved in the database. Dynamic content is difficult to cache because the loading time of the content is high. Dynamic pages are displayed after a complex compilation of the back-end issues, which is why using the cache system is essential [Raghunathan et al., 2011].

Based on the contemporary condition of a business, e-commerce applications are generated dynamically and stored in database systems. To enhance the response time and accelerate the content delivery, e-commerce web sites perform database query caching solutions [Hsiung et al., 2003]. Caching the data can be performed at several levels. In the web applications, the most common data caching happens on web browser of a user using proxy and database query cache systems [Sulaiman et al., 2008].

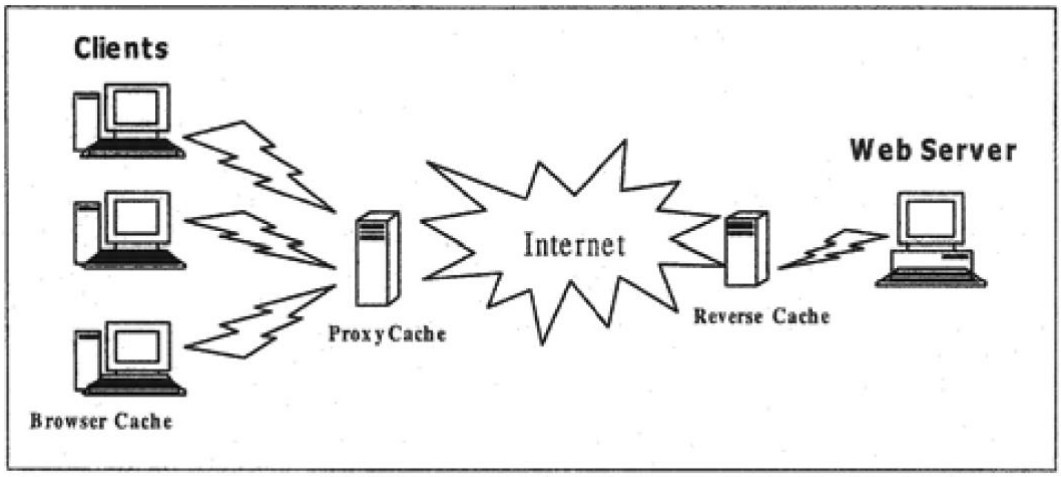

Figure 4. Web caching and prefetching at three cache levels [Wang 1999].

Figure 4 shows the three different levels in which web caching and prefetching can be implemented. The three cache levels for caching and prefetching include client side, proxy server and website. In the client, web caching can be implemented withing the browsers. The recommended caching is in the client's browser as the memory consumption is low. The benefit of sharing the browser cache objects is not shared with all the users as it is performed only in users' individual browser. The proxy cache server is the second level of web caching, and it is performed using the local hard disk on the gateway server for storage. Near the network gateways, the proxy caches are placed to reduce the bandwidth required over pricey Internet connections [Sulaiman et al., 2008].

3.1 Web application frameworks and web application quality

A complete web stack includes both the client-side and server-side technologies used to make a full-fledged application. The client-side technology includes HTML, CSS, JavaScript, and server-side technology includes, for example, PHP, ASP, Python, Ruby, or Node. LAMP (Linux, Apache, MySQL, PHP) was the first to invigorate as the tech Stack, defying a set of frameworks accounting for a complete web application, i.e., front-end (client-side) and back-end (server-side). JavaScript frameworks MEAN (Mongo DB, Angular JS, Express, and Node) and MERN (AngularJS replaced by React) are among the most used stacks. MEAN and MERN full-stack JavaScript -stack is formed around the Node.js cross-platform.

Ryan Dahl introduced a new kind of development environment called Node.js in the year 2009, which allowed JavaScript code to run server-side for the first time. The use of Node.js on the server-side simplifies, speeds up, and unifies the web-application development process because JavaScript language is used in all realms of the software. Traditionally there is client, server database architecture, which has a client application or web server delivers the user a

kind of a client application and the client sends a request or a message in which case the webserver needs data from the database that can be fetched and displayed.

Some web application frameworks provided a different model for web application organization, in which routes were more explicitly defined, decoupling them from a directory structure [Rodriguez et al., 2008]. Nowadays, the route manipulation approach is commonly used in modern Model-View-Controller (MVC) frameworks, such as Ruby on Rails. Using the best practices from other frameworks into one's own project is also considered beneficial, for instance, Scala (programming language) has a web framework called Lift that borrows the benefits from other frameworks while making the code simple and concise. Lift uses an effective template system based on the view-first approach, which makes the modification of pages and the reuse of page components much simpler [Salas-Zárate et al., 2014]. While choosing a framework, considering aspects such as separation of presentation, content, and logic, templating system, boot and scheme, and snippets is useful.

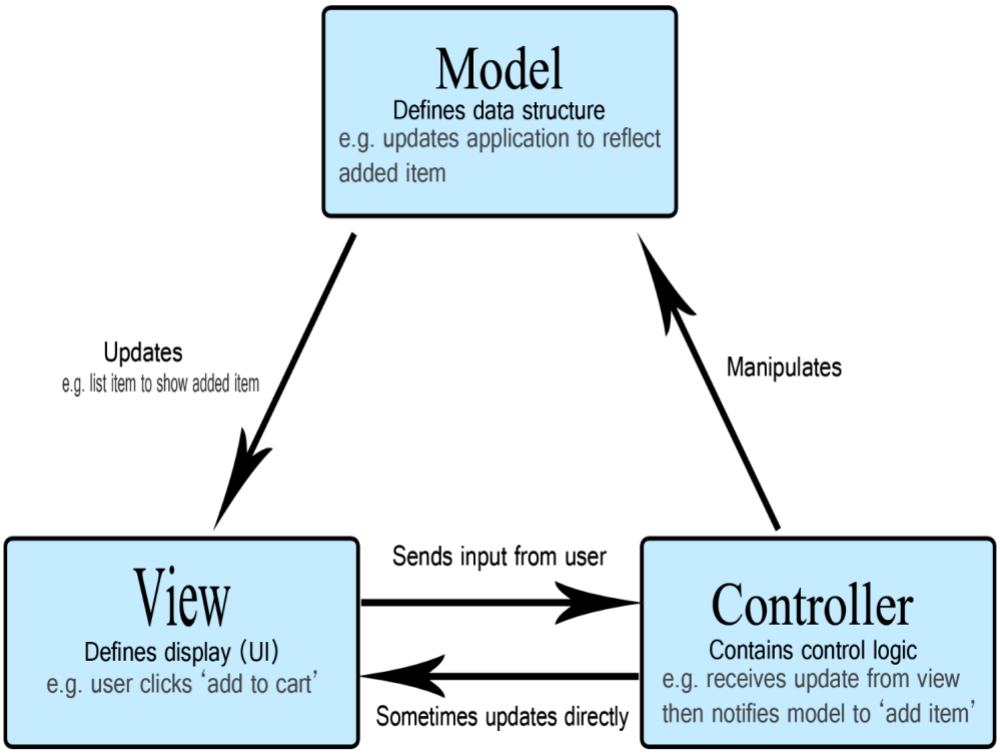

Figure 5. The architecture of a Framework [Mozilla Developer, MDN].

The basic structure of a web framework consists of the model, view, and controller as shown in Figure 5. The model supports the backend and contains all the data logic layers. The view takes care of how the page visually looks like (front end). The controller converts the input to commands [Pop et al., 2014]. A framework is a high-level solution built for the

reusability of software pieces. A web framework allows sharing the common functions and generic logic of a domain application [Salas-Zárate et al., 2014].

Developers should be aware of various aspects of the framework such as how to effectively build, structure, and organize Cake-based applications, and how to build more extensive web applications [Mavromoustakos et al., 2007]. Most web frameworks support features such as Internationalization, Forms Validation, AJAX support, and ORM, among others. However, some frameworks lack features such as Actors, Lazy Loading, Comet, Pattern Matching, HTML5, SiteMap, Wiring, and Parallel Rendering.

To gain reusability of the view and controller in the MVC pattern, the model class has to be defined with a set of types and should be constructed in such a way that the view components can easily access only required data information [Alpaev et al., 2007]. The MVC design pattern also has pros and cons that can be seen below.

Advantages of the MVC software design pattern are as follows

- Build multiple views for a model.
- Integrate with all popular JavaScript frameworks.
- Easy to change the view in UI (for example fonts, screen layouts, colors) and to add a new device without influencing the model.
- Return data using the same components with any interface.
- Supports web apps and SEO friendly web pages.

Disadvantages of the MVC software design pattern are as follows

- The high complexity of the application and structure.
- Difficult to debug the event-driven UI code.
- Not suitable for small applications that have an adverse effect on the application's performance and design.
- Could be difficult for novice developers to separate business logic and presentation tier.
- The isolated development process by UI authors, business logic authors, and controller authors may lead to delay in their respective module development.

Another kind of architecture is the three-tier organization, which is shown in Figure 5. It has three physical layers: client, application, and database. The database is usually RDBMS. The application communicates with the client using HTTP and also consists of the business logic running on a server. The HTML generated by the application layer is run by a web browser on which the client is working [Kals et al., 2006].

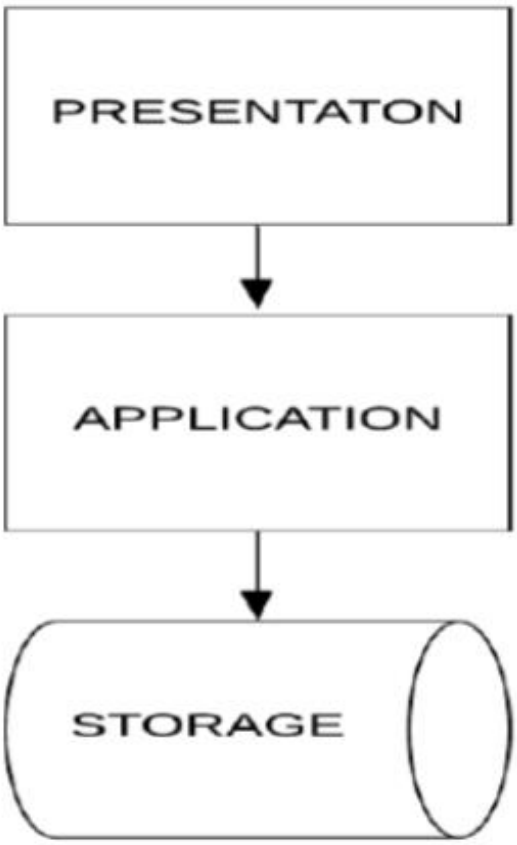

Figure 6. Three-tier architecture [Curie et al., 2019].

The three-tier architecture consists of three layers. The Presentation Tier is also known as the Client Tier. The Application Tier is for logical behavior, and the data Tier also is known as the data or storage Tier as shown in Figure 6. The Presentation Tier is having to do with how the application is shown in the browser, and as such it depends on the business model (business logic). The Application Tier communicates with the Presentation Tier and the data flow depends on what kind of data a Presentation Tier wants to request.

In the Application Tier, the authentication workflow takes place; hence the name Logic Tier. The storage layer is for the data that is stored, for example, it can be a SQL database storage, NoSQL like MongoDB, elastic search engine, or DynamoDB that enables the quick retrieval of the data [Caballe et al., 2014].

Once a web application is deployed, it is necessary to regulate if it continues to meet the requirements and expectations of the organization by regularly conducting quality assurance tests [Mavromoustakos et al., 2007]. The results of these tests are also utilized for further enhancing and improving the accuracy of the model by adding functions to serve new requirements or by modifying and enhancing the existing functionality [Mavromoustakos et al., 2007].

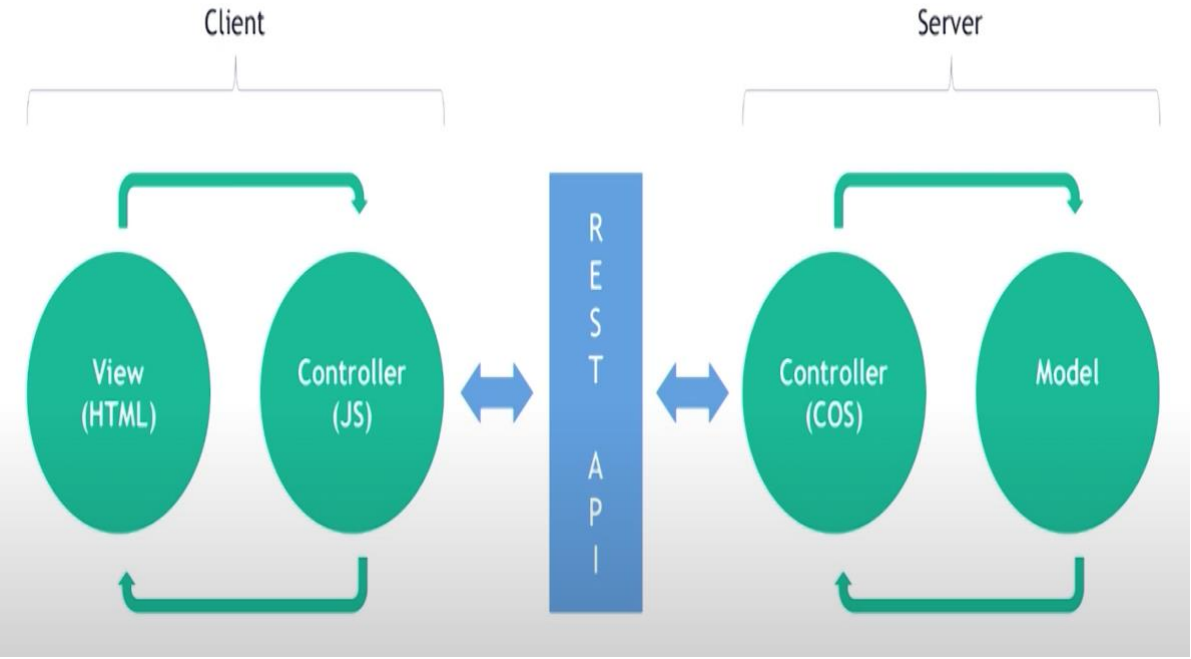

Figure 7. Client server and REST API [Malakhov et al., 2018].

Figure 7 shows the data flow between client and server with the REST API. In this approach, the controller is separated by the REST API, and it is possible to individually develop the web application. The Controller is separated by the REST API that is on the server. The benefit of using REST API in the application development is that both of the Controllers can be controlled separately. The incoming request from the Client is listened by the REST API once it is called. After that, the client-server data flow works same as in the MVC design pattern [Wilde et al., 2011].

3.2 Web application security

Once a web app is on the internet, it will be automatically scanned and attacked numerous times each day. So, if we look at the log files, multiple scanners are trying to do something bad. These are called web crawlers or web spiders or botnet. Anyone can access a website running on the server and can try to hack it by checking if the application is having some security issues. Web crawlers load the site and find out using the system if the site is running something that has some security issues. The information about the exact version of the webserver software, such as Apache, NGINX, Django, WordPress is running is enough for the attackers to hack a site. The hackers know the latest version of the web server software and if they find out that the site they are going to hack is still running on the earlier version and is not upgraded, they check on the internet what bugs were there in the earlier versions and how they were fixed [Fonseca et al., 2010].

Once the hackers get the information about which bugs were fixed by the company that has produced the webserver software, hackers can use various methods, such as Metasploit, Httrack, Black Widow, Website Ripper Copier, Burp Suite Spider to hack the site. The automatic tools that are freely available to download can be used by typing a command, which asks for the URL address that the attackers would like to attack [Kals et al., 2006].

The automatic hacking tools are user friendly and let the attackers enter the site by asking a few known vulnerability questions. If the attackers do not know the answers that a hacking software is asking, there is an option that lets attackers choose “I am not sure to try everything”, which then provides a list of ways to attack and tries to do all possible exploitation to the site in seconds. The job of an automatic attacker bot, or the hacking people is to try to imagine how to break the sites and prevent the developer's image to the ways to solve vulnerability issues.

A developer might just concentrate on the user interface and user-friendly aspects of a site. but then he or she is not aware of what the users might be doing. A normal user can also type certain SQL injection queries in the input field of a form and can easily cause a security problem. Even though the users are not aware of it while doing a simple attacking process, they can cause huge damage to the system and can easily break the site by spreading a virus or at least use the computer where the site runs to do some other malicious operations.

So, making a site available on the internet is inherently unsafe as it requires a lot of security work that needs to be taken into consideration. The use of the internet has a relatively separate logic compared to a normal software or application that runs on a personal desktop. While using a new framework to make a web application, it is important to run some security check commands for deployment.

Developers should do the strict configuration check-in development level before deploying to the production level. No matter how secure a website is, there are always people who are only interested in breaking the sites [Kals et al., 2006]. Among all the tests developers perform on web applications, security testing is perhaps the most important, yet it is often the most neglected [Hope et al., 2009].

Over 75% of network attacks are targeted at the web application layer [Cross, 2007]. A secure software is complex and depends on several factors, including good risk estimation, good code architecture, web server configuration, coding to prevent the most common attacks [Quinton, 2017]. It is always recommended for developers to run the freely available security tools at the development and production level and optimize the code as much as possible [Ramler et al., 2002].

4 INTRODUCING VALOS STACK

ValOS removes the complexity of software development and especially web development. No-code platforms that use drag and drop by generating the code or hook up components together and wiring services in sync behind the scenes. Valaa took a different approach by reforming the fundamentals of what software or web applications are, and how they are allocated and delivered to the users. One of the dominant things in the software space is to save applications as files and the Valaa platform does not use such an approach, so the traditional IDEs cannot be used. Instead, a web-based IDE named Zero is used to create the applications using Valaa. Valaa contains URM which takes the concept of a file and folder to combine with the concept of open class and an object in a way that all the basic features such as classes can have member properties, and it is possible to instantiate objects out from the classes. On the other hand, the storing can be done inside file folders.

ValOS property named “Lenses” are used to show the user interface to the users. The root VSX file generated while creating a Lens is similar to the index.html in web frameworks that acts as an entry point, which the server can detect as an entry point of the application. The keyword focus means to know as to what the focus of the Lens is so everything that is under the root entity of the project can be accessed with the focus keyword. The keyword focus lets us auto-update the changes in real-time in multiple tabs or windows of the browser.

ValOS Gateway is written entirely in JavaScript. The current Authority implementation is JavaScript as well. It utilizes several AWS services, including EC2, S3, Lambdas, DynamoDB, IoT Platform, and API Gateway. The first open-source Authority will most likely be written with JavaScript on top of Node.js. Tools and services within ValoSpace- like the Zero IDE - are built with ValOS itself [Kiiskinen, 2019].

4.1 ValOS concepts and terminology

ValoSpace is where every Valaa Resource is contained. Each Resource has a unique Valaa Space id, even if they are on different partitions or partition authorities. Valaa Resources are the basic building blocks of Valaa. These are Entities, Relations, Media, and Properties. Valaa Object Model is the concept through which ValoSpace is structured. A partition is a technically separated part of ValoSpace. A partition gets loaded separately from other partitions and is only loaded at the time when it is needed. They are used for optimization and access control. Since the platform is still under development, partitions also have formed their separate ValoSpace structure, but this is just a temporary restraint. Also, there is no access to control yet. Partition authority is where the partition data is stored. An Entity is one of the Resources, the basic building blocks of ValoSpace.

An Entity can contain other Resources and can have lenses with which it can be rendered. Media is one of the Resources, the basic building blocks of ValoSpace. Media is pretty much the same thing as a file - it contains data and has a type that determines how it is interpreted.

ValoScript Media is interpreted as code, while a PNG is interpreted as image data. Since the platform is still under development, the file extension of the name of the media is also used to determine the type.

The Relation is one of the Resources, the basic building blocks of ValoSpace. The relation has a target to another Resource and forms a two-way relationship between its owner and the target. The relation can also contain other Resources and lenses, just like Entity. Property is one of the Resources, the basic building blocks of ValoSpace: either a value ("literal") or a pointer ("reference"). The pointer is a property that has as its value a reference to another Resource and can be used to access that Resource.

Literal property is a property that has as its value some direct value like a string, number, or Boolean. An accessor is used as another name for a reference property when it has the same name and owner as its target is and used in a way where its meaning is just to provide VS access to the target (instead of, for example, designating a selected Resource). A Resource is usually "owned" by another Resource. The Resources owned by a Resource are its "Ownlings". If the owner is destroyed, all its Ownlings are destroyed as well, and if the owner is instanced all its Ownlings get instanced as well (as ghosts the instance is an explicitly created instance of a Resource). It can have its modification on top, while still following the prototype for the unmodified parts. An implicit inherited Resource created by an instance's prototype owning something is defined as a ghost. Due to the instance being equal to its prototype, any Resources the prototype owns will also appear under the instance. These are ghosts, and they act a lot like instances, in that they can be modified (becoming modified ghosts) and retain changed parts but follow their prototype on other parts.

Ghost instancing is the concept of an instance implicitly "having" its versions of the Resources its prototype has. These resources only "exist" when observed, so they don't leave any memory footprint. Yet, modifications can be made to them, and even then, all that is needs to be stored in the modification data. This is made possible by an underlying derived-id logic. An inheritor is something that, when observed, has the features of its prototype as if they were its own: instance and a ghost. Blob is the technical content of the media. This shouldn't be needed in normal usage. Whenever we edit the content of a Media, a new Blob that is the content is created, and the Media is changed to have a reference to that.

ValoScript is a language used to manipulate ValoSpace and is mostly compatible with JavaScript. Even though VS is run using the browser's JavaScript engine, the main difference is that the VS code gets interpreted through Valaa, which allows it have access to Valaa Resources and the context to manipulate them. It also has some standard methods accessible under "Valaa." VSX is a language similar to React's JSX, but instead of code being interpreted as JavaScript, it is interpreted as ValoScript. <ValoScope> is a VSX element that is used to render another Resource ("focus" of the ValoScope) that has a lens to render it, or another specific VSX. Focus is whatever Resource is "being rendered" using the VSX. Usually, this

would be the owner of the VSX. We use the name “focus” in VSX code when we want to manipulate the object, and we give <ValoScope> the property “focus” to determine which Resource we want to render. The context in this example can be used to get access to the individual array items because the focus is pointing to the whole array.

```
<ValoScope context={[origin:focus]} array={[“Kathmandu”, “Pokhara”,
“Syangja”]}>
<button onClick={() => origin.city = focus}>{focus}</button>
</ValoScope>

//focus.city would not work as focus is “Kathmandu”, “Pokhara”,
“Syangja”.
```

Code Example 1. Use of context.

In Code Example 1, context is making use of focus so that focus is no longer referring to the array items in the city array. The online editor named Zero offers several flexible options to build and deploy the project in a matter of time. All the entities and properties should be given an accessor property by right-clicking. The entities and properties that are given an accessor property right can then be accessed by other entities or properties, and they can be reused multiple times throughout the project.

The arrow icon on the left side of the entities indicates the element is given an accessor property right. Firstly, the root project itself must have an accessor property right given to it. Setting up a project in ValOS editor is easy and does not require any installation or in-depth programming knowledge for beginners to write simple codes. ValOS can be easily integrated into existing projects. Hovering over each element on the editor displays information for the developers as well as some suggestions that can guide developers as to what actions can be performed. The editor also has developer-friendly icons for each item.

ValOS replaces the very fundamental web concept of a static document with a dynamic event stream. It is not a framework, but a full-on reimagined approach to the core architecture of the Web [Kiiskinen, 2019].

ValoScript is significantly slower than regular JavaScript as it is transpired on the browser and converted to the intermediate query language VALK. We can still execute code as regular JavaScript by having a media someJavaScriptCode.js with the export default function () {console.log("hello!")} and running it in ValoScript as:

```
    const script = someJavaScriptCode.$V.immediateContent({mime:
    “application/javascript”})
    script()
```

Code Example 2. JavaScript code running in ValoScript (VS).

Code Example 2 shows how to use JS inside a VS code in ValOS. It is important to think of one ValOS partition as the equivalent of the traditional web document [Kiiskinen, 2019]. The partition is a single unit of information in terms of loading and security (read, write, etc.). The more is in one partition (event stream), the longer it takes to load. More partitions, longer those take to load. In the future there is going to be a feature called Query Authorities, which are PHP-equivalent of ValOS, which for example generate virtual event logs from more efficient database systems. Security ValOS has multiple layers of current and upcoming security features. Cryptographic content security including end-to-end encryption, is in progress within the EU funded project: General data security is handled using a permission system which is almost production ready [Kiiskinen, 2019].

4.2 Debugging

ValOS event streams are stored in DynamoDB as sequences (one sequence per partition, e.g. event stream). The same data can be found and inspected with Chrome dev tools by

F12 -> Application -> IndexedDB -> <PARTITION ID> -> truths

Most browsers should be supported, although there are issues with Microsoft Edge due to differently implemented IndexedDB API. Blobs, e.g. binary data objects are stored in AWS S3 and referred to by their hashes in ValOS Media's [ValOS.content] field.

```
console.log("Relation count now: ",
this[Valaa.relations].length);
```

Code Example 3. Console log.

It is possible to open the developer tools and console and use console.log directly from the VS script as shown in Code Example 3. However, console.log can handle and display a lot more than the window alert. It is also worth remembering console.warn() and console.error(). ValoSpace modifications that the execution already made will not get persisted since throwing an exception aborts the transaction.

Debugging in VSX

In VSX, we can use console.log directly:

```
<div>Hello. {console.log("Meow is", focus.meow)}</div>
```

Code Example 4. Using console.log inside VSX.

Code Example 4 shows how to use the login to the console by writing the code in VSX. Note that this may get triggered multiple times, due to how the rendering system works.

Debugging with Zero

Using Zero's "Run VS here" feature, open a quick box named "VsBOX" to run VS targeting the required resource. Note that it is also possible to do ValoSpace operations from the VSBox, not just console.log and other debugging.

Debugging with property edits

```
Const debug = Valaa.Resource.tryActiveResource("DEBUG_ID_HERE");
if (debug) debug.foo = this.foo;
```

Code Example 5. Debugging with property edits.

Code Example 5 shows that another way to debug is to have a process set a specific property under a "debug" entity, which can be looked at with Zero. We can target that entity easily with Valaa.Resource.tryActiveResource(id). We could combine this with the Promise trick while working on a local partition if we want to debug to a cloud partition:

```
const debug = Valaa.Resource.tryActiveResource("DEBUG_ID_HERE");
if (debug) {
      Promise.resolve().then( () => {
      debug. foo = this. foo;
      });
}
```

Code Example 6. Promise resolve on the local partition.

To get the id of the debug resource, do the "Run VS here" and click on the name displayed on the VSBox. Code Example 6 shows how the promise resolve works on the local partition.

Debugging Resources in general

Direct console.log of a resource is usually not that informative, and it is much easier to debug resources by using foo[Valaa.name] and foo[Valaa.rawId].substring(0,8).

```
console.log(foo ? foo[Valaa.name]+"/"+ foo[Valaa.rawId] : foo);
```

Code Example 7. Resources debugging.

However, these will fail when foo is not a valid Resource, so it is good to use the ternary operator to safeguard them as shown in Code Example 7.

```
console.log(foo?foo[Valaa.hasInterface]("Resource"))?
foo[Valaa.name]+"/"+ foo[Valaa.rawId] : foo : foo);
```

Code Example 8. Debug truthy value.

Code Example 8 shows how to debug a truthy value. However, if there is a chance that the resource is truthy but just not a resource, Code Example 8 can be used.

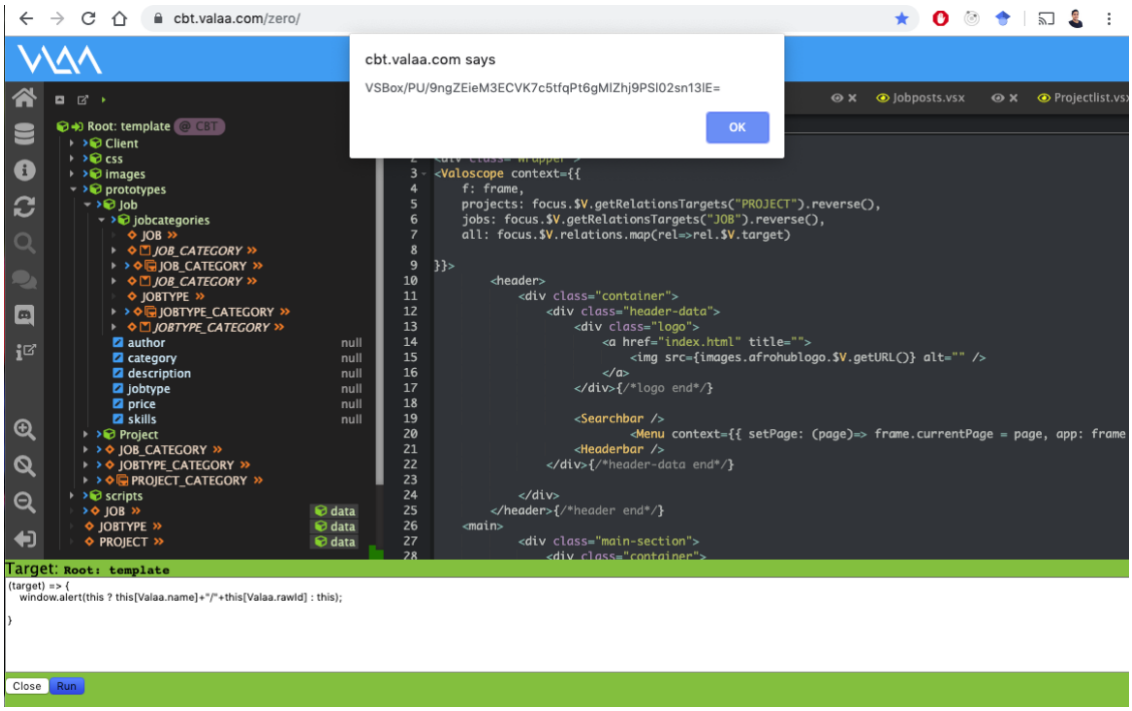

Figure 8. Debugging to see the unique raw Id of the root template project.

Figure 8 shows the example code to debug and alert the unique raw ID of the root template project. The debugging code is written in VS.

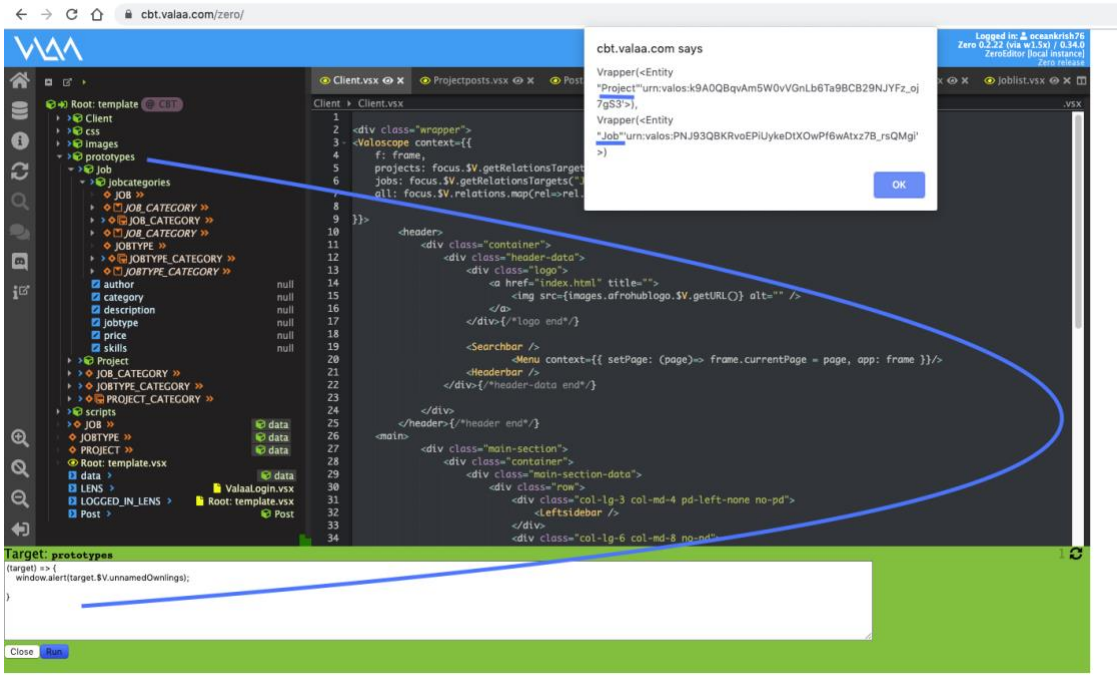

Figure 9. Debugging to see what the prototype resource owns.

Figure 9 shows how to check the entities or resources that are owned by a particular prototype resource.

```
<h1> {typeof focus} </h1> can be resource
<ValoScope array={focus}{typeof focus}>
{Array.isArray(focus)? "is array": "is not an array"}
```

Code Example 9. Display if the focus is an array or not.

To check whether the focus is an array, object, or some other data type, a type of syntax can be used just like in JavaScript as shown in Code Example 9. One issue with this is that it shows focus also as an array even if it is an object. For that we have to check using this code: {typeof focus === "object"? "is object": "is not an object"}.

4.3 ValoScript (VS)

ValoScript extends ECMAScript 5 object model transparently for manipulating ValoSpace resources. The ValoScript interpreter creates events from all ValoSpace resource modification side-effects and groups all such side effects into transactions. ValoScript retains ECMAScript 5 syntax and semantics [Kiiskinen, 2019].

```
new Entity({
   name: "the name",
   owner: theOwner,
});
new Relation({
   name: "the name",
   Owner: theOwner,
   target: theTarget,
});
new Entity({
   name: "the name",
   owner: theOwner,
   properties: {
          someFoo: 3
   },
});
```

Code Example 10. Entity and Relations Resources in ValOS

Code Example 10 shows how to create Valaa Resources for example Entity, Relation instances with the VS code. The Relation can have its target pointed to some other entity. Entity and Relation can both have properties as well. The entity and the relation can have the same owner.

```
this. foo = "boo"; // create a property foo with value boo
const moo = new Entity({ // create entity, store it to variable
  name: "moo",
  owner: this
});
this.activeThing = moo; // make a pointer property
this.moo = moo; // make an accessor property
moo.species = "cow";
console.log(pointerMoo.moo); // "cow"
this.myCow = moo;
console.log(this.myCow.moo) // "cow"
```

Code Example 11. Creating properties and log to console.

Code Example 11 shows how to create regular properties as well as the pointer and accessor properties with VS.

Functions and their usage

```
[Valaa.hasInterface](interfaceName)
E.g. [Valaa.hasInterface]("Entity")
// Returns an array of incoming relations of “foo”.
[Valaa.getIncomingRelations]("foo")
// Returns an array of relations named “foo”.
[Valaa.getRelations]("foo")
// Returns an array of sources that are in relations named "foo".
[Relatable.getIncomingRelationsSources]("foo")
// Get target objects of relation.
[Relatable.getRelationsTargets]("foo")
```

Code Example 12. Get the resource by its raw id.

To get a resource by its raw id const, Code Example 12 can be used. However, the resource should be active, which means it needs to be on a partition that is already loaded. With the given example code 12, we can check if Resource has a specific interface and it can be used for example to check whether it is an Entity, Relation, or a Media.

```
() => {const newMap = "<div>Hello world</div>";}
```

Code Example 13. VSX in VS.

Generating VSX Media files from VS is made easier in ValOS and can be achieved by using Code Example 13.

Focus

A good practice is to have a focus on the Client and not change it. Whenever the content needs to be displayed using focus, create <If test={}> <Post /> for it. So, to show the content with a button that changes using frame.showProfile == true and then hide the content by making the frame.showProfile == false with a button.

Frame

The frame is always only used in one component/entity. Because of this, we could use two same-named frames inside client and menu entities, and they would not know about each other. In another entity, context can be used while calling the Menu component.

```
<ValoScope
array={focus.$V.getRelationsTargets("JOB").reverse().slice(0, 1)}>
<h6 style={{color: "#000000"}}>{focus.title}</h6>
<p>March 27, 2019</p>
</ValoScope>

{focus.title[0]} // Gives the first character from the title.
```

Code Example 14. Displaying the last element from a relation named “JOB”.

The last element from a relation “JOB” can be displayed by using the code as shown in Code Example 14.

```
<input type="text" onKeyUp={(e) => {
if(e.which === 13) {
        new Relation({
            owner: focus,//targets to public user
            name: "message",
            properties: {
                message: e.target.value
            }})
        e.target.value = null;
    }}
} />
<ul><Message array={focus.$V.getRelations("message")} /></ul>
```

Code Example 15. Chat application using ValOS.

Code example 15 shows the entire chat application using ValOS with just a few lines of code. The code as shown in Code Example 15 can be saved as Chat.vsx and can display as a list by writing {focus.message}, in another entity Message.vsx.

4.4 ValoSpace- the ValOS resource model

ValOS applications both manipulate and consist of ValOS Resources. There are three types: Entity, Relation and Media, which can all have Properties. These Resources are unique, and their state is hosted by *authorities*. Together they conceptually form a global resource space that is called *ValoSpace*.

ValOS applications are stored as Resources, and they change the state of other Resources. The Resources are hosted by *authorities*. To access the ValoSpace resources and run ValOS applications, we need a software called a *gateway.* Gateway is the "client" or "engine" that runs the application and interacts with the authorities.

The current user gateway ("inspire") runs in a web browser and can run ValoScript (VS) code to manipulate the resources. VS is a flavor of JS, most ES2016/ES2017 features are supported [Kiiskinen, 2019]. The gateway can display ValOS resources in the browser DOM, described as VSX (similar to React's JSX). VSX is XHTML that allows using curly braces to embed VS code and ValOS resource references.

The browser downloads the gateway (inspire-gateway.js and plugins) and configuration (revela.json). The configuration has the (VALOS) URI of the (ValOS) application to load. The URI consists of a protocol, an authority, and a partition on that authority. The gateway connects to the authority using a specified protocol, fetches the application, and starts running it. Running an application means rendering the partition's specified default *lens*. A lens is just a single VSX Media ("file"). The application displays things to the user as "normal" browser dom. The gateway keeps in memory the state of the application's corner of ValoSpace (any partitions it has been requested to load). When the application makes a manipulation to its local state (also conceptually part of ValoSpace), the gateway alters the DOM to match the changes.

An application very likely accesses data on different partitions. As it does so, the gateway transparently notices this "request" to data, pauses application activity, connects to the requested authority/partition, fetches the data, and continues application activity. It subscribes to further changes using MQTT.

On the application layer, this access is transparent - the ValOS application does not need to care where the data resides, just contain a reference to wherever it happens to be. The gateway does the rest. Manipulating local state is the same as manipulating cloud-stored data; for the application, there is just the global ValoSpace state, and it is the gateway that handles cases differently depending on where that state is stored. Only the part of the application using unloaded data is paused. If the unloaded data is expected to be displayed in the UI, an alternate view (lens) can be defined to be displayed temporarily during loading.

4.5 Data modification and synchronization

When the application attempts to make a change to the state, the gateway immediately changes its internal state, and triggers relevant changes in the rendered UI, regardless of where that state is stored in. But in case the data is hosted on a remote authority, it then sends the same change as a *command* to the authority that hosts that state. Depending on the access rights, the authority may accept or reject the change.

If the change is accepted, the authority then sends the change as an *event* to all gateways that are using the data (subscribed to MQTT events). The gateway that originally sent the command has already made the change, so it just marks it as accepted. However, for any other listening gateway, this is a new thing, and they will now make the same change to their internal state, causing a trigger of relevant changes in the rendered UI.

If the authority responds with rejection, the gateway will rewind its state to what it was before rejection. If it has already made other changes afterward, it will then try to reapply those. Once this internal rollback and replay of the state are done (in milliseconds) it will trigger relevant changes in the rendered UI.

This all happens transparently to the application (unless the developer deliberately hooks up to the lifecycle). The application code simply modifies the data, and any of its UI displaying that data will reflect any changes whatever they end up being. Authorities don't usually care much of the *state* they store changes, but they do make sure there is a consistent log of changes to it. All events in an authority are sequential, and each command to change it contains information on what was the last event. A conflict happens if there is a mismatch - the authority rejects the event, which causes the gateway to fetch all the latest events to catch up with the authority. It is then up to the gateway to resolve this, which usually causes the attempted change to just be reapplied on top of the remote changes, and then sent again to the authority.

ValOS gateway provides two special authorities called ValOS-local and ValOS-memory which are private to the gateway itself. Since ValoSpace content is hosted by authorities, and the application layer doesn't care which authority the state is hosted on, manipulating this local state works the same as manipulating cloud data. Any ValoSpace resources hosted on ValOS-memory will be discarded on browser refresh and for ValOS-local the gateway will persist the changes, but they will only be accessible locally by the gateway (i.e. clearing indexedDB will destroy all local state in a browser, and an incognito window will have a separate local state cleared on browser exit).

4.6 ValoSpace instancing and ghosts

Any ValoSpace resources can work as prototypes for other (instance) resources. If the prototype owns other resources, these will appear as *ghosts* on the instance. Any manipulation on these ghosts will be contained to the specific instance, and since the system uses event sourcing. There is no limit to the depth of this *ghost instancing*. When we instance the root

entity, we can access “ghosts” underneath it, and under its ghosts, at any depth, and any changes to a specific ghost in the structure will be stored as a single change. Instances can also reside on different partitions and authorities than their prototypes. This makes it extremely easy for ValoSpace applications and application parts to function by altering their state. Rendering a resource as an *instance lens* will create an in-memory instance, effectively containing all its structure as if it would be in memory, while also being a very low-cost operation since it practically only creates one resource. Thus, the application or application part can alter its behavior by simply editing any state it has, with changes contained in the gateway running the application.

As a general rule, the authority stores state changes as events, while the gateway processes the logic. An authority is comparatively much closer to a database than a backend. ValOS doesn't make much of a distinction between a frontend and backend - both are just gateways running ValOS application code, backend just usually has privileged access to data partitions and better access to the client system than a browser. Backend (called a "worker") has fetched the gateway code from npm (@ValOS/inspire, @ValOS/toolset-worker and plugins) and has the configuration (toolsets.json). The configuration has the URI of the application to run, and a process is started (CLI: VLM perspire) to run fetch and run that code.

Since the backend gateway is running with a node, not within a browser, it can have access to any parts of the host environment that are provided to it. Those parts can then be directly used from the ValoSpace application code written in VS. Since the backend code is also part of ValoSpace, it can (and usually is) stored in a cloud authority and edited using a normal browser gateway (that has been authenticated to access and modify it). Authorities generally store events as logs of events. When these are played back (by the gateway), they reduce into a specific ValoSpace state. An authority will generally contain multiple event logs - these are called *partitions*. Each partition is its atomic event log and will always be accessed as a whole. Whether the partitions are hosted on the same single authority or multiple different ones doesn't matter to the application.

A ValOS application, and especially its data, will generally consist of multiple smaller partitions, spreading its state into multiple event logs. From the ValoSpace perspective, it does not matter how the application is split up into partitions - for the application any ValoSpace Resources are identical regardless of where their state is stored. If an onclick handler creates several relations, the transaction gets initiated by the root call from the UI. So, upon clicking a button, it creates a new transaction. The transaction then gets committed only when the whole function call chain comes back out. If an onclick handler creates several relations, all of the relationships go through the same transaction. To isolate the created relations, the use of promise is needed so the callback of the promises can have their transaction.

If the ValoSpace state resulting from the event log gets changes that make earlier changes obsolete, it is possible to compress the event log into a *snapshot*. In this case, the authority will

deliver only the latest snapshot and changes applied after that. This way the gateway doesn't need to process the entire history to reach the state, only a relevant representation of it. A simple example shows the following events: alpha=1; beta=3; alpha=8. This event log could be compressed into a snapshot: beta=3; alpha=8. These both result in the same state {alpha: 8, beta: 3} - since the state remains the same, the snapshots only work as architectural optimization and thus can be created at any point without affecting ValoSpace.

Since partitions are always fetched as a whole, the structure becomes relevant when access rights are considered. An authority may allow a user to only narrate (read) the event log of a specific partition, but both add to (write) and narrate another one. There can be more detailed access rights for write operations, but for reading the partition, the gateway needs to have the entire event log to construct a valid ValoSpace state. The access rights are determined by ValoSpace resources called access relations, which state the desired access rights. The authority will read modifications to these relations and act accordingly.

Identity is based on ownership of an *identity partition*. Authentication to current authorities is done using a side-channel: an identity provider with a trusted channel to the authority will contain information on who owns a specific identity partition, and on a successful challenge-response (e.g. password) will provide a token to both the user and authority. The gateway will deliver this token along with any requests to the authority. Authentication depends on the specific authority implementation, and thus there are numerous possible alternate models for authentication. However, certificate-based authentication for infrastructure level identity verification and event signing is also in the works.

5 NETWORKING WEB APPLICATION WITH VALOS

ValOS is a rethinking of how we handle web applications and their data. The concepts that modern web tech is built on are based on patches on top of an outdated model, which leads to unnecessary complexity. Unnecessary complexity causes hard-to-manage large systems and hinders the entry to development. ValOS systematically breaks down the fundamental concept of using microservices in projects. Unlike many stacks these days, ValOS makes use of a light-weight process of constructing multiple projects and sharing the data between them for all kinds of projects from small to large.

ValOS is based on everything being dynamic from the ground up. This is achieved with distributed event sourcing. There are two layers: state and the changes. ValOS applications are built from, and deal only with the state, and can completely ignore the changes. The underlying architecture handles the layer of changes [Kiiskinen, 2019].

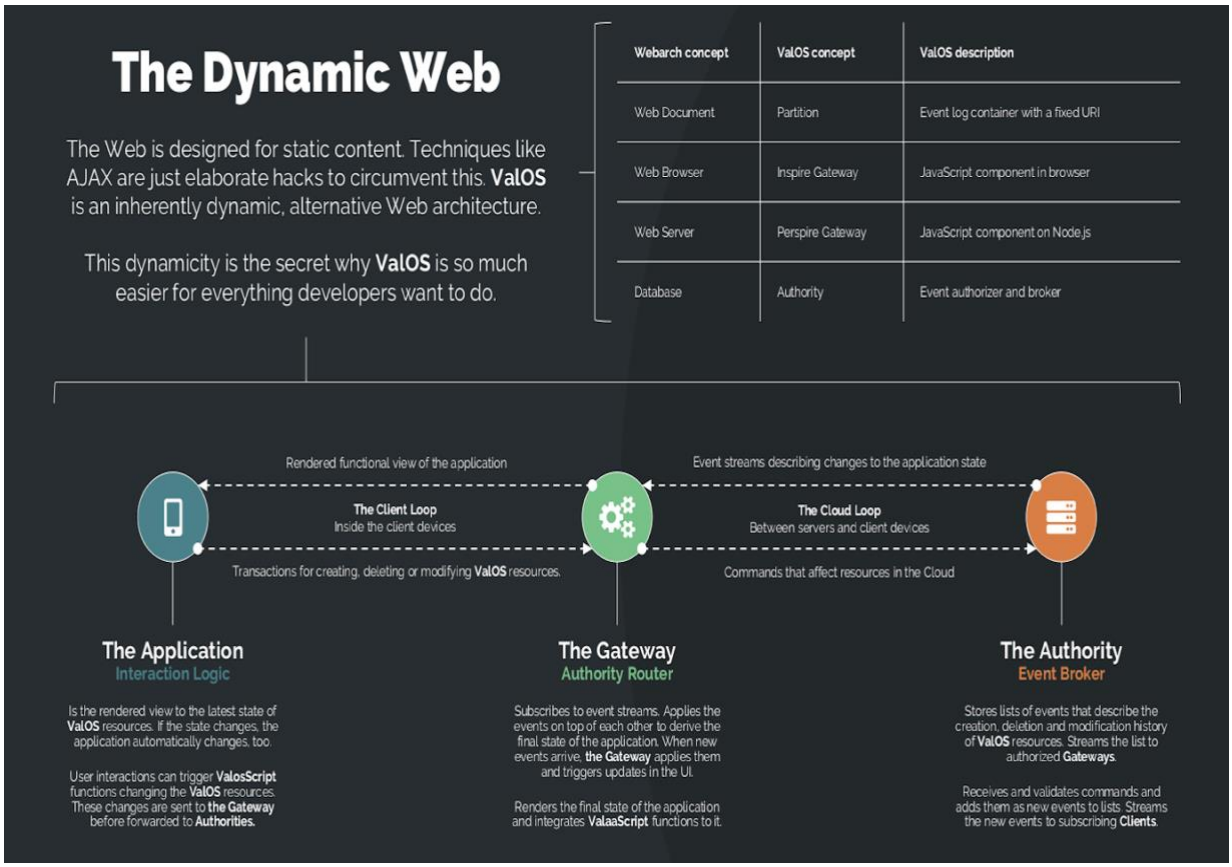

Figure 10. The Dynamic Web model [Kiiskinen, 2019].

The fundamental change in ValOS with the traditional web architecture as shown in Figure 10 is that ValOS swaps the places of the webserver and the database. Instead of storing software as files and state as kind of database entries, Valaa uses the technique of storing applications as a series of events. Essentially the event sourcing technique covers not only the application state, but the application itself. The IDE emits the events to a database, or a list called partitions, while developing the application.

The partitions are stored in a cloud called the authority, which is an event broker tasked to store, validate, and distribute events. In the browser, there is a library called gateway, which is an open-source library that Valaa Technologies is building and is the core and heart of the ValOS Loop system. The gateway is capable of subscribing to event streams and reducing the application state out of the event stream. Once the gateway gets the event streams, it renders the functional view of the application to the user so whatever is displayed in the browser is the rendering of the final state of the event stream from the gateway.

At the same time, the application integrates ValoScript functions to the user interface which are capable of creating transactions that describe the creation, deletion, and modification of the resources back to the gateway. If the transaction is affecting other resources in the cloud, it sends that as a command to the authority, which validates the event, so it checks the authentication of the users who have the permission to make the changes.

Once the validation process is completed by the authority or the server and it passes the validating phase, the events are then sent back to the gateway. If the gateway was the original creator, nothing happens other than it is checked by the authority and it is validated. The resources stored in the cloud are being looked at using the component named ValoScope, which contains features such as Lens that describes how the resources are looking and focus to know exactly where is where the ValoScope pointed at.

5.1 The project structures

The structure of the project has been changing constantly as the development process has evolved. Some of the data in the project structure might not be sown as they are partitioned and handled separately in the background for security reasons. The iterative Agile Software Development pattern helped shape the structure and outline of the project as it evolved. The structure of the project is quite simple and provides easy access to all of its elements in a categorized manner.

The Networking web application built with ValOS is for people of all ages. Social networking means the use of Internet-based social media sites to keep contact with family, friends, colleagues, and customers. Social networking can have a social purpose, a business purpose, or both, through sites such as Facebook, Twitter, LinkedIn, and Instagram, among others. The networking web application built with ValOS includes features such as creating a user account, offering and seeking information, getting matched with the people that share at least three common interests, uploading profile photos as well as making posts.

In this project the data contains the following entities:

- Adder: Adds the user posts.
- Interests: Stores the interests from the user input and assigns the interests depending on the category they select.
- Prototypes: Interests and tags can inherit the properties and methods from prototypes.

- Tags: Store the tag categories that can be chosen by the users when they add to their profile.
- Users: Store a list of users and the properties that point to their respective entities.

The symbols in the entities, references, instances, properties and root entity are well organized as shown in Figure 10, which gives a developer an understanding of the project structure at first sight in Zero online editor.

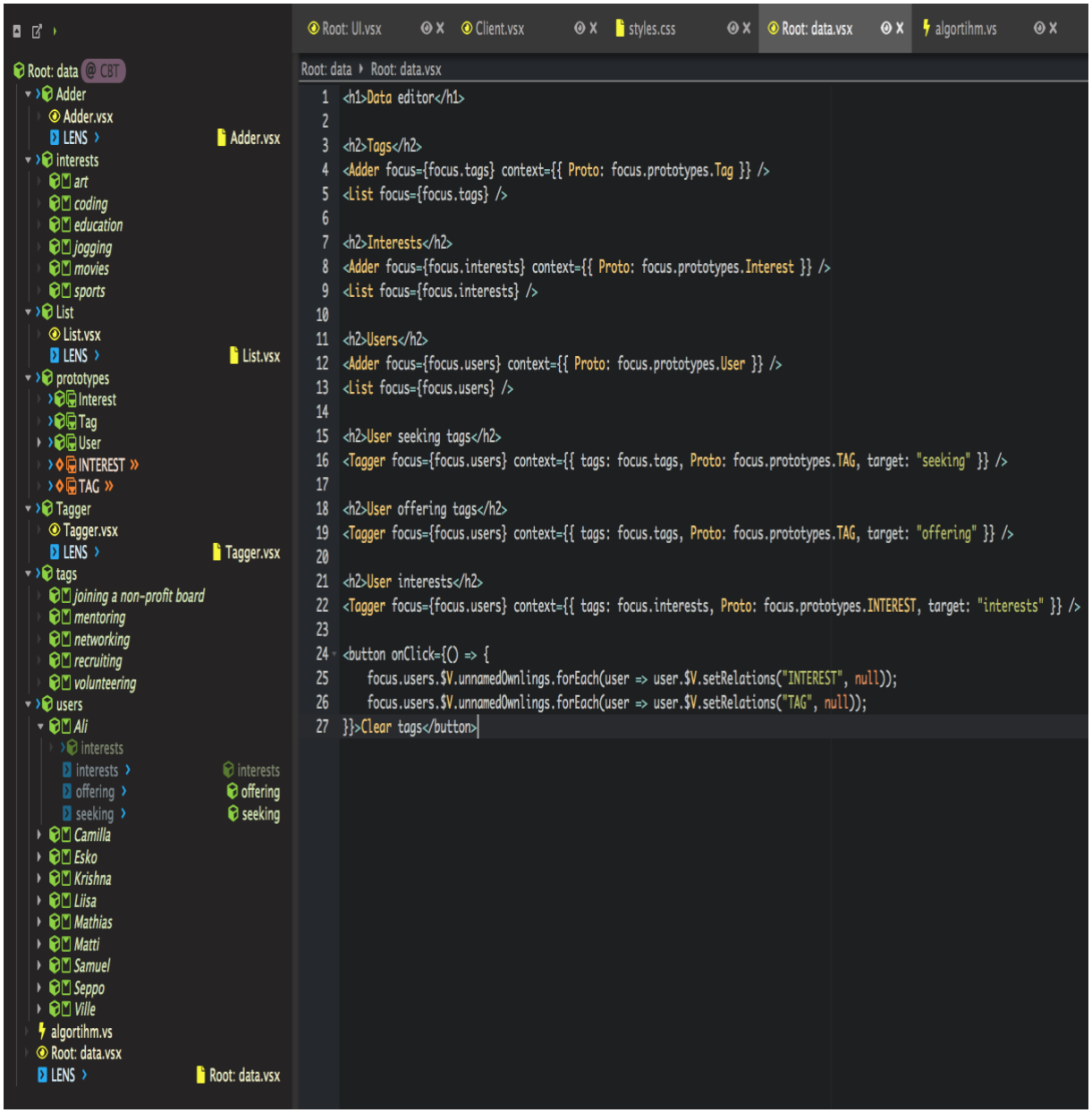

Figure 11. The Project Structure and the root file names “data.vsx”.

Figure 11 shows the project structure of the Networking web Application on the ValOS platform. There are two main entities under a root project named data and UI. Data is used for storing and manipulating the user input data and the algorithm, whereas UI is used for the view purpose of the application.

5.2 Networking and matchmaking algorithm

The basic idea in matching the users or the organizations with one another is by calculating the scores, which are determined by the common tags and interests they share. The scoring is calculated to round up in the percentage, the higher score generated will be considered 100%, and the rest of the scores are arranged in a range of 0% to 100% in consideration with the score values that can be any numbers but are round up to range by making use of the highest score of 100%.

client ▸ algorithm.vs

```
(rels, me) => {
    let relations = rels.map(p => p.profile).filter(p => p.visible);

    if(!me) {
        return relations
            .map(p => {
                return {
                    profile: p
                };
            });
    } else {

        return relations
        .map(p => {
            let score = 0;
            const X1 = me.seeking.$V.getRelationsTargets("TAG");
            const Y1 = me.offering.$V.getRelationsTargets("TAG");
            const X2 = p.seeking.$V.getRelationsTargets("TAG");
            const Y2 = p.offering.$V.getRelationsTargets("TAG");

            const I1 = me.interests.$V.getRelationsTargets("INTEREST");
            const I2 = p.interests.$V.getRelationsTargets("INTEREST");

            const P1 = X1.filter(t => Y2.includes(t));
            const P2 = X2.filter(t => Y1.includes(t));
            const P3 = I1.filter(i => I2.includes(i));
            const P4 = I2.filter(i => I1.includes(i));

            let IR1 = 0;
            let IR2 = 0;

            if(I1.length !== 0) IR1 = P3.length / I1.length;
            if(I2.length !== 0) IR2 = P3.length / I2.length;

            score = (((P1.length + IR1) / 2) + ((P2.length + IR2) / 2)) / 2;

            return {
                profile: p,
                score: score
            };
        })
        .sort((a, b) => b.score - a.score)
        .map((p, index, profiles) => {
            p.percent = Math.round((p.score / profiles[0].score) * 100) || 0;
            return p;
        });

    }
};
```

Code Example 12. Matchmaking Algorithm algorithm.vs.

Code Example 12 shows the algorithm used for matching the users that are seeking something with the users that are offering something, for example, common interests such as sports or working together on a project. The code was written in VS.

JS matchmakingalgorithm.js

Users > Demola > Desktop > Thesis > JS matchmakingalgorithm.js > scored > users.map() callback

```
const me = {
    seeking: ['job'],
    offering: ['networking'],
    interests: ['sports', 'music', 'a', 'b', 'c', 'd', 'e', 'f']
};
const a = {
    seeking: ['mentors'],
    offering: [],
    interests: ['sports'],
    name: 'a'
};
const b = {
    seeking: ['networking'],
    offering: ['job'],
    interests: ['music'],
    name: 'b'
};
const c = {
    seeking: [],
    offering: [],
    interests: ['sports', 'music', 'a', 'b', 'c', 'd', 'e', 'f'],
    name: 'c'
};
const users = [a, b, c];

const scored = users.map(user => {
    let score = 0;
    const X1 = me.seeking; // job
    const Y1 = me.offering; // networking
    const X2 = user.seeking; // a = mentors, b = networking
    const Y2 = user.offering; // a = [], b = job

    const I1 = me.interests; // sports
    const I2 = user.interests; // a = sports, b = music

    const P1 = X1.filter(t => Y2.includes(t)); // a = job, b = []
    const P2 = X2.filter(t => Y1.includes(t)); // a = [], b = networking
    const P3 = I1.filter(i => I2.includes(i)); // a = sports, b = []
    const P4 = I2.filter(i => I1.includes(i)); // a = sports, b = []

    let IR1 = 0;
    let IR2 = 0;
    //CALCULATING INTERESTS see commons
    if (I1.length !== 0) IR1 = P3.length / I1.length; // a = 1/1 = 1, b = 0/1 = 0
    if (I2.length !== 0) IR2 = P3.length / I2.length; // a = 1/1 = 1, b = 0/1 = 0

    score = (((P1.length + IR1) / 2) + ((P2.length + IR2) / 2)) / 2;
    // a = (((1 + 1) / 2) + ((0 + 1) / 2)) / 2;
    // a = (((1 + 1) / 2) + ((0 + 1) / 2)) / 2; = (3/2)/2 = 5/2 = 0.75

    return {
        user: user,
        score: score
    }
})
    .sort((a, b) => b.score - a.score)
    .map((p, index, profiles) => {
        p.percent = Math.round((p.score / profiles[0].score) * 100) || 0;

        return p;
    });

//console.log(scored.slice(0, 1));
console.log(scored);

// node matchmaking.js
```

Code Example 13. Matchmaking Algorithm algorithm.js

The code given in Code Example 13 translates to Vanilla JavaScript correspondingly from the VS in ValOS. The code shown in example 17 is written in JS and can be run separately with node algorithm.js.

5.3 User interface

The basic design is functional and works well with multiple screen widths and is compatible with various devices. Making a fancy user-interface was not a priority in this project, but to be able to display the data in a simple view. When the data components are steadier and do not require revision, the priority will shift to the user-interface. In this project, the term user-interface mostly refers to the basic outlook of the website, such as the appearance of the button, menu bar, div, icons, texture, typography, and background. The overall visual representation of the web-view is thought to be user-friendly if the equal distribution of the information is emphasized and balanced. However, colors, sizes, textures, positionings, and, focuses are taken into consideration.

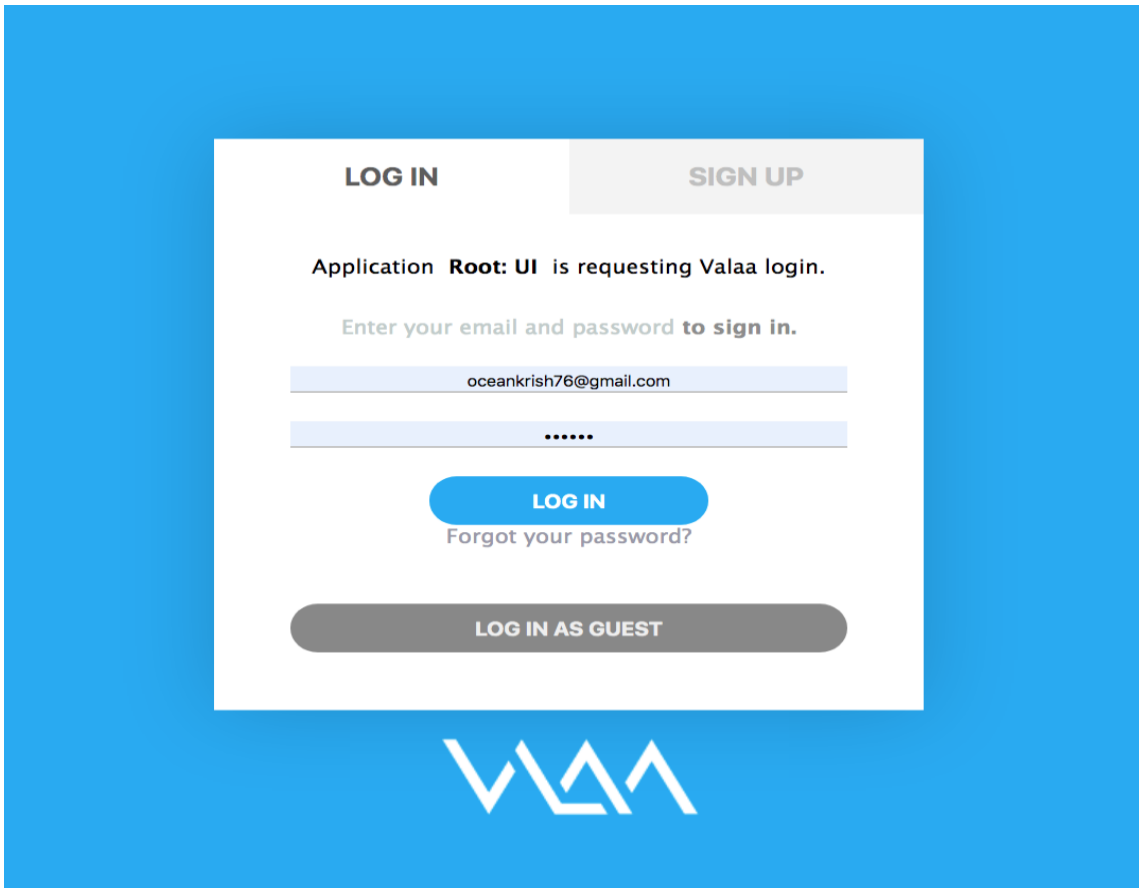

Figure 12. Valaa's login page.

The user-interface as shown in Figure 12 is Valaa's service for any basic users to get started with Valaa's user authentication system. Only after having the basic Valaa account, the users can create their profile. This makes the matchmaking service more secure and user-friendly

even though the process seems longer in the beginning. Any texts and profile images that are shown in the user's profile are associated with Valaa's basic authentication account.

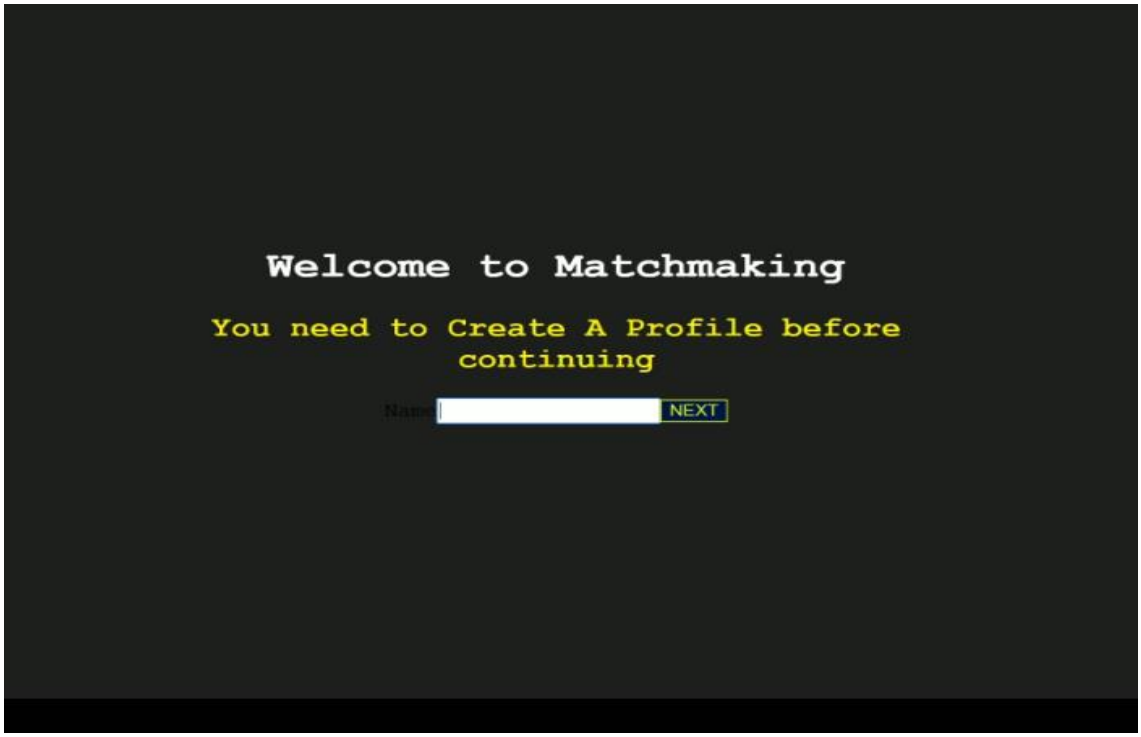

Figure 13. Creating a profile and inserting additional data (Interests, Seeking, Offering).

After logging in with Valaa's basic authentication system, users are asked to create their profile to use this matchmaking service. Figure 13 shows the user interface for creating the profile that asks for the profile name. After pressing next, it navigates to another user-interface where the users are asked to fill up their interests, the services they offer, and the services they seek. The interests, seeking tags and offering tags can be written with comma-separated words, which are then processed, saved, and allocated as each entity in the ValOS system.

Each item is allocated as an entity resource in the ValOS system and can have their separate properties and relations. The development part describes the way of processing data, how the comma-separated words are transformed into a separate entity and saved in the system. The user input fields for all the required data and the overall user-interface is kept simple and is easy to use so that any novice user can make use of the provided service. The users are guided with the textual representation. Creating profile names, inserting interests, seeking and offering data can be done on the same page, but it can be a little confusing for the new users, which is why the pages are available by pressing the next button after each insert. Only after creating the profile, the users are navigated to the main matchmaking page and can see other users. This increases the security and the users can also go to the settings page if they want to modify, add or delete their information.

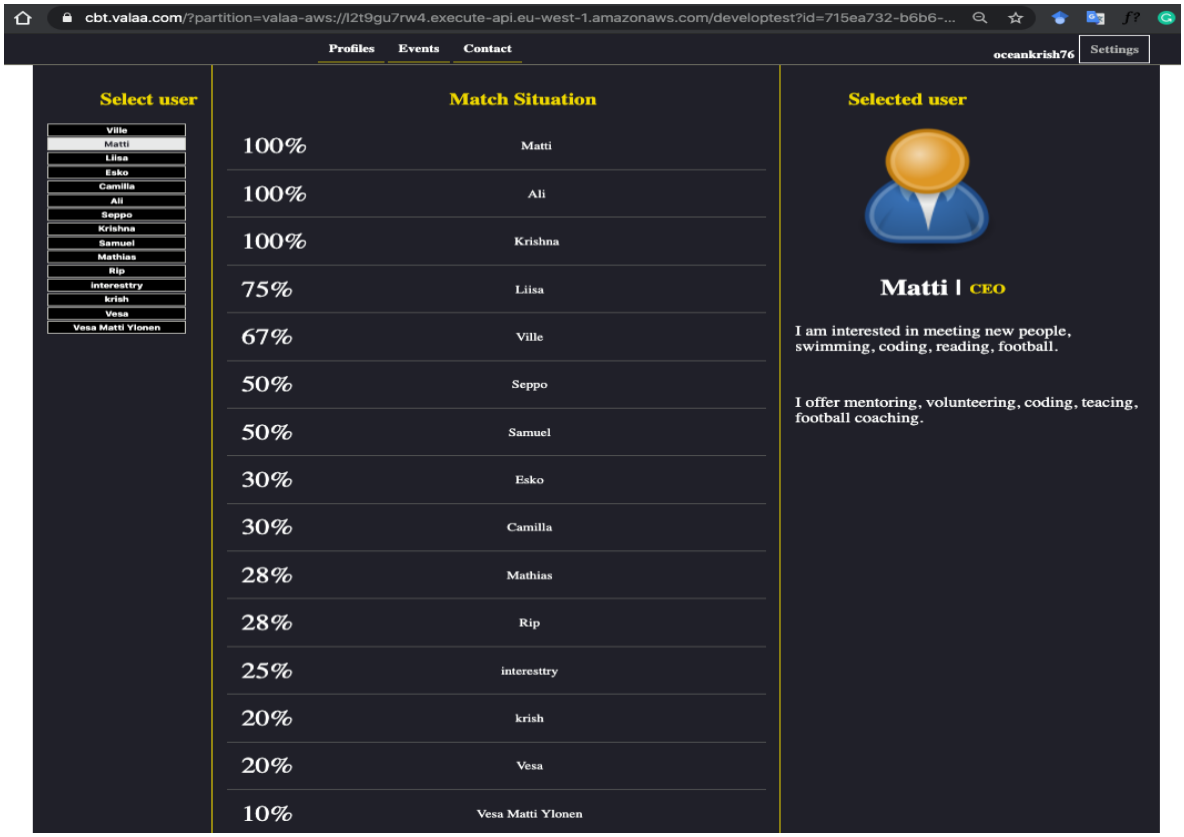

Figure 14. User-Interface of the main page.

Figure 14 shows the main page of the application after a successful login, where the users are shown their matching percentage with other respective users. The matching percentages are calculated depending on the match of the profiles. For example, selected user Matti as shown in Figure 14 has one hundred percent match with Ali, because both of the users share the same interests, seeking and offering tags. After the UX analysis, there are a few adjustments to redesign the User Interface of the Networking app such as reconstructing the main screen, editing the navigation bar, and reorganizing similar kinds of functions. However, these are just assumptions. The research needs to continue to expose the users' opinions and the data flow should be smooth before investing the UI part.

When we make changes to the code in Valaa Zero and we go offline or to a place where the network is poor, the static contents that were changed earlier do not appear on the site. The solution can be to make use of the service workers and at some point, there is going to be native ValOS service workers. Service workers let the app load faster on subsequent visits in production and gives offline capabilities. However, it also means that developers (and users) will only see deployed updates on subsequent visits to a page after all the existing tabs open on the page that have been closed since previously, cached and the resources are updated in the background.

5.4 Comparison between ValOS and MeteorJS

When we load a react page, at some point the component gets loaded, and when we close the browser the component gets destroyed. Whereas in Valaa, the component is persisted in the event stream so when someone loads a browser, we get the clone of the current state in the browser environment. However, it cannot be called as component lifecycle because there is a concept of the component at the server level. In React, there is no concept of the component at the server level as we have to have data in the database.

The server does not necessarily mean that there should be something constantly running in it. The data model is like a stream of events. the lifecycle of the component is much more persistent. In React, there is a strict division between code and the data. In ValOS there is no strict difference between the content and the code. So, when we build a system, we build the code and the content at the same time.

However, in React we have code somewhere and we create some mock data to create the components. The mock data can be used in the production system. Only after the mock data is created, we start to create the real data in React. React is like a lens of ValOS, not from the developer's point of view. React is an open-source library that ValOS can also use, but we can't write react on the ValOS system. ValOS creates its own DOM. So, the dependency on React is more technical. When or if we want to get rid of the dependency issues, ValOS is a perfect replacement for React. ValOS is not related to React other than in components being reused.

After severe research, personal experience, and mentor's advice, the comparison concepts between ValOS and MeteorJS are listed as programming languages, latency, HTTP request and response methods, full-stack reactivity, front-end, back-end, database, client-side cache, web sockets, loading time, auto-update and seamlessness. For a practical understanding, implementing a simple project on either of the compared technologies, ValOS or MeteorJS is recommended. The main reason to compare between ValOS and MeteorJS is that they share some similarities and the MeteorJS framework is found to be closely related with the ValOS stack even though the MeteorJS is not a stack onto itself unlike ValOS, which is the whole stack package.

ValOS stack and MeteorJS bring unique concepts to web application development and worth looking for people who are in the intriguing field of the mobile, web application, and SPA. Even though the ValOS is a web stack, the developers are free to use any software design pattern, for example, MVC, and most importantly, the developers are not required to make a separate back-end logic to communicate with the database. However, to have a full-stack web application using MeteorJS, developers are required to make database configuration by themselves and they have to write code for the back-end logic by themselves. Hence, developers who are familiar with the MERN stack or any other stack are believed to grasp the concepts relatively faster.

Table 2. Comparison between ValOS and MeteorJS.

Concept	**ValOS**	**MeteorJS**
Languages	Vs, Vsx, Js	Js, Node.js
Latency compensation	This has to do with databases. ValOS deals with relatively small event streams describing changes to a "thing". The front-end of an app might be one or several chronicles (event streams), but a complex application can have hundreds or thousands of chronicles. New events modify one chronicle at a time, so there is very little latency (as there are no database operations). In "vanilla" ValOS, the entire database (e.g. the resources) is loaded. This is an important difference between MetoerJS and others vs. ValOS in that ValOS does not include nor does it replace databases, it is just making web documents a lot more useful.	Meteor snapshots the individual data documents without freezing the contents of the entire database on every method call and updates the client-side. Data changes to other documents are applied directly without being buffered.
Request/response	Uses HTTP to fetch the initial event payload and then MQTT to receive updates from an authority, uses HTTP to send commands to authorities (which then respond with a validated event).	Makes use of the "connect" module, which is an HTTP library.

In Table 2, the important concepts such as the language used, the time factor, and request and response protocol between the client-server communication are compared. ValOS uses the VS, VSX, and JS programming languages and MeteorJS uses JS and Node.js for the back-end process. Since the database operation is not performed in the event-streaming concept with ValOS, the latency compensation is relatively low as compared to MeteorJS. However, in very method calls the database is not affected as MeteorJS snapshots the individual data documents [Robinson et al., 2015]. MeteorJS uses a cache to unmodified data on every call, which helps in processing the data changes faster. HTTP method fetches the initial payload and response events are verified by the authority, which uses MQTT to send the updates back in ValOS.

MeteorJS uses the “connect” HTTP library for receiving and sending the request and response messages.

Table 3. Comparison between ValOS and MeteorJS.

Concept	**ValOS**	**MeteorJS**
Full-stack reactivity	In ValOS, every property is wrapped to a thing called LiveKuery. They listen to events that affect them and then update UIs and perform computation. It also has two modes: immediate state mode and transactional mode. Immediate mode is enabled if obtain Discourse is not specified. In this mode, the subscription value and listener notifications are based on the internal getState() of this subscription. refresh state will fetch the state from the currently active top-level engine.discourse otherwise transactional mode is enabled.	Tracker provides reactivity on the front-end. The tracker allows automatic templates to rerun and other computations whenever session variables, database queries, and other data sources change.
Front-end	The Front-end of ValOS is delivered with Inspire. VSX is the user interface language of ValOS. ValOS can be used as a backend that exposes REST APIs that can be then used with any JavaScript front-end.	Blaze library is used, and alternatives can be also used such as ReactJS or AngularJS.
Back-end	ValOS has a very unique way of doing back-end code with Inspire running on Node.js. Effectively it has a virtual DOM that reacts to events just like the front-end does and then performs computation. Perspire can be extended with a web-spindle, an HTTP server capable of exposing rest APIs to ValOS [Kiiskinen 2019].	For backend in MeteorJS, the Apolo server is used. GraphQL was used to practice a prototype application for APIs and routing for the comparison propose.
Database	Data is stored in the AWS cloud in DynamoDB as lists of events that describe changes to resources.	Data is stored in MongoDB.

Table 4. Comparison between ValOS and MeteorJS.

Concept	**ValOS**	**MeteorJS**
Client-side cache	ValOS has a client-side cache based on IndexedDB.	The client does not have to wait for a roundtrip to update the user's screen. Whenever the database changes to the server, it simultaneously updates its local cache with its requested changes and also makes changes in the real-server-side.
Web sockets	ValOS uses WebSockets for the MQTT connection.	Meteor subscribes to the outcome of a real-time query so that any currently generated or modified documents that are matched will be sent to the clients over fast WebSockets connections.
Time factor/Loading	Since ValOS uses the query language VALK after on a browser transpiling the regular JS, it is relatively slower.	The use of a single language makes it relatively faster to load the content.
Auto-update	In ValOS there is no strict difference between the content and the code. So, when we build a system, we build the code and the content at the same time. The component lifecycle is persistent as the data model is based on the stream of events.	A strict division between the code and the data exists.
Seamlessness	No self-installation of any package is required. Both the server-side and client-side offer asynchronous auto-updates.	Installation of Meteor required. Also has the auto-update feature.

Table 3 shows the comparison between the ValOS stack and MeteorJS framework with the comparison concepts full-stack reactivity, front-end, back-end, and database. The fronted of ValOS uses Inspire and VSX as the user-interface language similar to JSX in ReactJS. Whereas, MeteorJS uses the Blaze library. Blaze is a powerful library for creating user

interfaces by writing reactive HTML templates. Familiar template directives like {{#if}} and {{#each}} integrates with Tracker's "transparent reactivity" and Minimongo's database cursors so that the DOM updates automatically. Table 4 compares the difference between ValOS and MeteorJS with the concepts, client-side cache, web sockets, loading, auto-update functionality, and seamlessness. Besides the comparisons listed in Table 4, ValOS allows rapid prototyping by cutting off the setup process. However, with MeteorJS, creating a base configuration is required and after that, the application development process is faster. ValOS is commonly used for SPA.

5.5 Project evaluation

The overall development of the application for this project "Networking Web Application with ValOS stack" was executed effectively. The requirements of the features in this project have changed a lot during the project development lifecycle. The frequent change in the requirements developed some difficulties in the beginning and also during the development phase. For example, the push notification system was planned to implement in the application so that the users could get useful notifications. Nevertheless, the push notification system could require a computational server-side component that is more complex than the generic Authority. ValOS has a system called the Perspire Worker – it is effectively a Node.js -based server-side Gateway that can react to changes in ValOS resources, create new events, and do generic operations and tasks any Node.js server can do. The problem with this is that it is significantly harder to develop, poorly documented, requires maintenance, monitoring and the developer would have to get into AWS development. Additionally, it would cost between 40-70 dollars a month as the work requires an entire EC2 node. Most likely these features would create such overhead for the company, that it is not possible to get the support for a small project.

Although there were a few limitations on the project scope, most of the project requirements were accomplished, and the final result is satisfying. The application was developed, and it worked perfectly in the ValOS system and was also able to transfer to my domain. The practice of the DSR research method and the AR method was effective as the thesis was the outcome of the "build and evaluate" as well as "learning by doing" process. The combination of the two research methods made the development process vibrant as actively planning, designing, committing, observing, evaluating, and repeating the process because of the requirement changes and frequently occurred errors solved a lot of issues right away. The evaluating of the sprints and each task was carried out together with the Valaa's mentor and clients. Dealing directly with the clients and manifesting their ideas into reality was conducive, which was achieved with the frequent guidance of the mentor. A lot of times, the tasks and the whole project itself had to restart from the scratch. That is a positive and iterative way of perfecting the required tasks. The DSR approach aimed primarily at discovery and problem-

solving as opposed to the accumulation of theoretical knowledge. The bridging between the DSR and AR methodologies was explored in which the problem-solving, iteration practice was the main intersection and they both complemented one another.

Table 5. The functional requirements of the project.

Requirements	**Information**	**Implementation**
Creating profiles	Must: Permission controlled profiles with edit, update, and delete operations. Could: The email validation is less so and requires another server-side component (to send the emails). Must: Show key information quickly on page load.	Implemented.
List of profiles	Must: Display a list of profiles with quick information and match percentages on the main page.	Implemented.
Poster/images	Must: Profile images and icons	Implemented.
Push notifications	Could: Feature to send useful push notifications to the users.	Not implemented.
Match Percentage	Must: People sharing the number of same interests, seeking and offering tags must be displayed from 1-100%.	Implemented.
Messaging system	Must: Match(s) who have a 100% match percentage have to be able to message each other.	Implemented.
Events creating	Must: Create events and invite matches.	Implemented.
Users on map	Must: Display matches on a map.	Not implemented.

Table 5 lists the basic functional requirements and analyses the requirement concepts as well as the final column shows whether the requirement is implemented or not depending on the circumstances during the entire software development lifecycle. Side by side, trending web stacks, frameworks, and libraries such as MERN stack, MeteorJS framework, and ReactJS library were learned as part of hobby and for the comparison reason, which gave a lot of new skills and experiences. Software project management skills, teamwork, and non-IT knowledge such as time management and keeping minutes were acquired. Bug fixing and error handling

techniques were also acquired. Since Valaa Technologies has its unique system in the whole internet and web application development world, the articles from the external resources were not as helpful for the literature review of this thesis. However, there are many similar articles on the internet that are copies of each other. Reading other articles opened different points of view and provided useful background information. The results are evaluated and analyzed of the extent to which the project met its intended goals, and the functional requirements were delivered.

5.6 Developer experience

The setting up for the application development process was faster as Valaa's platform is straight forward and the mentor was helping in all the necessary processes to kick start the project. The thesis supervisor was also interested in the unique solutions for the modern-day internet services that Valaa Technology is offering and helped the author in shaping the thesis writing. Due to the developer's previous unfamiliarity with some of the unique solutions, integration, and documentation that Valaa Technologies have, it took a relatively long time to grasp the intentions of some of the methodologies. However, also due to the developer's previous experiences in web application development and a great interest in modern-day solutions, the overall experience was thoroughly special. Although there are many similarities between some of the programming languages with the VS and VSX, there are different syntax and unique approaches, which took a long time to get used to them. The terms such as focus, frame, the count was extremely confusing in the beginning and continued to be confusing for a long time.

While developing the networking web-application with ValOS and concurrently observing the development and thesis writing process, AR methodology increased productivity. The constructive and functional suggestions were obtained along the development process, which resulted in increasing the pace and also resolving the immediate issues and changes. A lot of time was spent reading the documentation and perform small tests and develop small projects such as counter, chat system, fetch API, CRUD operations, handle user input data, and display the necessary view in the browser. It was easy to ask the mentor how to implement certain methods to achieve the desired UI or manipulate data back and forth. Automated testing tools were not used during the development and the errors were confusing in the beginning. Therefore, there was a need to manually ensure that the written code was running correctly. However, the errors were clear and displaying immediately right in the browser, which made it easier to go back to the editor and make the changes accordingly.

Valaa Technologies has provided the developer with the necessary resources, tools, and guidance to achieve the requirements of the project, which is to build a real-time full-fledged web application. It was a great experience to be in presence with the people from Valaa Technologies who have been in this technology field for several years and have produced

ground-breaking products that entice the developers and business entrepreneurs. Throughout the development process, all the concerned parties were equally excited for this thesis writing and supported in all necessary realms and for that, the author is forever indebted and grateful. The consistency of the author was fluctuating because of the side projects and other studies. Features such as email validation technique, push notifications can be implemented and if some company or a person wants to have this project for themselves, the application can be extending with additional features and delivered upon agreement with Valaa Technologies. It is hoped that the main problems will be avoided. These instructions are not binding, they are intended to save writers from dwelling on details of mode of presentation. Styling of the application is minimal as intended, but Bootstrap 4 is already integrated with the ValOS system so there is a freedom of using any features from Bootstrap to render a fancy view for the end-users. It is also possible to integrate the ValOS with ReactJS, make use of Material-UI so that the user-friendly fractures from those libraries and framework can be easily accessed.

The benefit of using a framework is that it speeds up the development process. MongoDB is relatively easier to implement with the MeteorJS framework than the relational database, for example MySQL. Because MeteorJS uses the Blaze.js and Flow-router, it is considered to be easier for developers who are familiar with JS. ValOS stack and MeteorJS framework contribute largely to the web application development process. For a novice developer, it might be difficult to develop a product with fewer faults and might have to dwell in the configuration process as well. Since JS is a popular programming language, JS frameworks are evolving continuously to bring the full-fledged SPA, which is rendered fast. There are a wide range of JS frameworks and relatively few full-stack JS. Therefore, ValOS popularity is believed to be increasing over some time. Besides, Blaze, Meteor's frontend rendering system users are switching to ReactJS as Blaze is a Meteor-only package. The constant switching of the developers between one framework to another is common these days. A MeteorJS application is an amalgamation of JS that runs inside a client web browser and JS that runs on the Meteor server inside a Node.js container and supports HTML fragments, CSS rules, and static assets.

Saving applications as file is the most common and dominating in the software development system. ValOS does not save the applications as a file, which means that the traditional IDEs are not used in creating the applications in ValOS as their focus is on file systems. In ValOS, the unified resource model takes a concept of file and folder and combines with the concept of class and objects in a way that classes can have member properties and object instantiating. For a beginner, learning a basic "todo" application is the starting point. With ValOS it is possible to create the entire "todo" CRUD application with just 17 lines of code and the data is stored in the database as well as the real-time changes appear immediately on all browsers. For anyone who has written JSX in ReactJS can easily grasp the concept of VSX in ValOS.

6 CONCLUSION

The internet service and web software are an intriguing field, and a lot of changes and transformations have happened for ten-twenty years. New stacks, frameworks, and libraries are created with the basic foundation that helps developers save time and focus on the core of the software development process instead of dwelling on the setup processes. The developers do not need to sit for hours just to get the configuration working and mostly the compatibility and accessibility issues are solved in a speck of time. A wide range of tools is providing the high performance of the software system. The developer's community is working together to develop the solutions every time that increase the performance and scalability. It was found that most uprising stacks, frameworks, libraries, or similar platforms have some shared characteristics and elements within their architecture, which makes them both developer-friendly and rapid for development.

ValOS also shares similar characteristics with the modern-day web frameworks and mostly with MeteorJS. MeteorJS was found to be one of the most active frameworks among all the discovered frameworks and compared during the thesis. Since ValOS contained almost all fundamental components of the modern web architecture and a unique solution of its own, the development resulted in an easy and straightforward experience, compared to the development with other modern-day stacks such as MERN and MEAN. This thesis presents sufficient evidence to change the way developers think of both front-end and back-end development. ValOS reduces the work on creating boilerplate and having to create classes and files. ValOS can use the libraries such as ReactJS, but it is not possible to write React to the ValOS system. ValOS creates its own DOM. So, the dependency on React is more technical. When or if we want to get rid of the dependency issues, ValOS is a perfect replacement for ReactJS.

ValOS is not related to ReactJS other than in components being reused. When we load a React page, at some point the component gets loaded when we close the browser the component gets destroyed. The benefits of ValOS data model architecture to increase the overall development speed and persistency, as well as the dynamic web model have been studied. The advantages of partitioning and manipulation the data between the ValOS entity or components were listed and tested in practice. The developed networking web application on the ValOS system that was built for the practical implementation in this thesis project demonstrates the simplicity of data flow achieved with the ValOS dynamic web model. During the development of a networking web application process, it was discovered that despite having a persistent and real-time with a user-friendly model of the ValOS, the page load and performance issues are lacking compared to the demo projects from its competitors. There are many factors to consider when choosing a well-equipped tool for starting up. As far as the question of choosing a framework/stack/library is concerned, we should analyze our needs and targets. This relies much on the work's requirements, and occasionally your current skillsets.

Although many companies have chosen web framework as an appropriate technique to aid its development process and enhance coding standards, most frameworks are very strict. Sometimes, we may practically find it easier to code by hand and using a framework might be even more complicated. The list of advantages and disadvantages of the framework over coding by hand may prove helpful when making the decision.

Advantages:

- Already established foundation and skeleton to get going.
- Reusability, compatibility, and accessibility advantages.
- Cost-effective and timesaving.
- Dynamic content rendering and database connections are easier.
- Automate common web development tasks such as caching, session authentication, routing, and URL mapping.

Disadvantages:

- Programmers choosing frameworks before digging deep into the core language's problem-solving skills.
- Lack of option to modify the core behavior.
- Speed and the performance of the application.
- Small applications where the developers need to have total control over the code.
- Collaboration issues if the teammates cannot afford to learn a new framework.

Mostly in small web applications, the use of web frameworks is not necessary and can even cause a lot of damages. Because of the ever-growing e-commerce and large commercial companies wanting to have the database and users to create and modify the content, the popularity of the web frameworks is sky-high. The choosing of the web frameworks also depends on the factors such as the developer's learning curve, the framework's available documentation, a large community of supportive community, frequent updates and bug fixes, performance, adaptability, compatibility issues, and the market values.

7 REFERENCES

Alpaev, Sergey (2006). Applied MVC Patterns. A pattern language. Dnepropetrovsk, Ukraine; arXiv preprint cs/0605020.

Cross, Michael & Fisher, Matt (2007). Developer's guide to web application security. Rockland, MA: Syngress Publishing.

Dutta, Kaushik & Datta, Anindya & Vander Meer, Debra & Thomas, Helen & Ramamritham, Krithivasan (2007). ReDAL: An Efficient and Practical Request Distribution Technique for Application Server Clusters. IEEE Transactions on Parallel and Distributed Systems 18(11):1516–1528.

Edge Jr., Charles S. & Barker, Chris & Schwiebert, Ehren (2010). Web Servers. In book Beginning Mac OS X Snow Leopard Server from Solo Install to Enterprise Integration. 1st ed. Berkeley, CA: Apress, 357–396.

Fonseca, José & Vieira, Marco & Madeira, Henrique (2010). The Web Attacker Perspective - A Field Study. 299-308. 10.1109/ISSRE.2010.21.

Hevner, Alan R. & March, Salvatore T. & Park, Jinsoo & Ram, Sudha (2004). Design Science in Information Systems Research. Management Information Systems Quarterly, 28 (1), 75–105.

Holmström, Jan & Ketokivi, Mikko & Hameri, Ari-Pekka (2009). Bridging Practice and Theory: A Design Science Approach. Decision Sciences, 40(1), 65–87.

Hope, Paco & Walther, Ben (2009). Web Security Testing Cookbook (1st ed.). Beijing: O'Reilly Media, Inc.

Hsiung, Wang-Pin & Li, Wen-Syan & Candan, K. Selcuk & Agrawal, Divyakant (2003). Multi-tiered Cache Management for E-Commerce Web Sites. In: Chan Alvin & Chan, Stephen & Leong, Hong Va & Ng, Vincent (eds) Cooperative Internet Computing. The Springer International Series in Engineering and Computer Science, vol 729. Springer, Boston, MA.

Iivari, Juhani & Venable, John (2009). Action research and design science research - Seemingly similar but decisively dissimilar. 17th European Conference on Information Systems, ECIS 2009. 1642-1653.

Iskandar, Taufan & Lubis, Muharman & Kusumasari, Tien & Lubis, Arif (2020). Comparison between client-side and server-side rendering in web development. IOP Conference Series. Materials Science and Engineering, 801, 12136–.

Iyengar, Arun & Challenger, Jim (2000). Improving Web Server Performance by Caching Dynamic Data. USENIX Symposium on Internet Technologies and Systems, 49–60.

Kals, Stefan & Kirda, Engin & Krügel, Christopher & Jovanovic, Nenad (2006). SecuBat: A Web vulnerability scanner. Proceedings of the 15th International Conference on World Wide Web, 247–256.

Kiiskinen, Iridian (2019). ValOS introduction and ValoSpace API reference. Licensed under a Creative Commons Attribution 4.0 License, 0.35.0-35.

Kulesza, Raoni & de Sousa, Marcelo Fernandes & Araújo, Matheus & Araújo, Claudiomar & Filho, Aguinaldo (2020). 'Evolution of Web Systems Architectures: A Roadmap', in Special Topics in Multimedia, IoT and Web Technologies. Cham: Springer International Publishing, 3–21.

Madasu, Vamsi Krishna & Eltaeib, Tarik (2015). Web authentication and authorization and Role of HTTP, HTTPS Protocol in Networking. An article on the ResearchGate website.

Malakhov, Kyrylo & Kurgaev, Oleksandr & Velychko, Vitalii (2018). Modern RESTful API DLs and frameworks for RESTful web services API schema modeling, documenting, visualizing. Problems in programming 2018; 4: 59–68.

Mavromoustakos, Stephanos & Andreou, Andreas (2007). WAQE: A Web Application Quality Evaluation model. International Journal of Web Engineering and Technology, 3(1), 96–120.

Offermann, Philipp & Levina, Olga & Schönherr, Marten & Bub, Udo (2009). Outline of a design science research process. Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, 1–11.

Quinton, Eric (2017) Safety of Web Applications: Risks, Encryption, and Handling Vulnerabilities with PHP. San Diego: Elsevier.

Raghunathan, Aravamudhan & Murugesan, K. (2011) Performance-Enhanced Caching Scheme for Web Clusters for Dynamic Content. International Journal of Business Data Communications and Networking (IJBDCN). 7 (3), 16–36.

Ramler, Rudolf & Weippl, Edgar & Winterer, Mario & Schwinger, Wieland & Altmann, Josef (2002). A quality-driven approach to web testing. Proceedings of Ibero American Conference on Web Engineering 2002, 81–95.

Robinson, Josh & Gray, Aaron & Titarenco, David (2015) 'Getting Started with Meteor', in Introducing Meteor. Berkeley, CA: Apress, 27–41.

Rodriguez, Alex (2008). Restful web services: The basics. IBM developerWorks, 33, 18.

Salas-Zárate, María del Pilar & Alor-Hernández, Giner & Valencia-García, Rafael & Rodríguez-Mazahua, Lisbeth & Rodríguez-González, Alejandro & López Cuadrado, José Luis (2015). Analyzing best practices on Web development frameworks: The lift approach. Science of Computer Programming, 102, 1–19.

Shklar, Leon & Rosen, Richard (2009). Web application architecture: principles, protocols and practices. 2nd ed. Wiley.

Stephen, Fleming (2014). System and web security agent method for certificate authority reputation enforcement, US2014101442 (A1).

Sulaiman, Sarina & Shamsuddin, Siti Mariyam & Abraham, Ajith & Sulaiman, Shahida (2008). Web caching and prefetching: What, why, and how? Proceedings of International Symposium on Information Technology 2008, vol. 4, IEEE, 2008, pp. 1–8.

Tripp, David (2005). Pesquisa-ação: uma introdução metodológica Action research: a methodological introduction. Educação e Pesquisa, 31(3), 443–466.

Van Aken, Joan Ernst (2005). Management Research as a Design Science: Articulating the Research Products of Mode 2 Knowledge Production in Management. British Journal of Management, 16(1), 19–36.

Wang, Jia (1999). A survey of Web Caching Schemes for the Internet. Computer communication review. 29 (5), 36–46.

Wilde, Erik & Pautasso, Cesare (2011). REST: From Research to Practice. New York, NY: Springer.