LIVEMACHE: SUPPORTING COLLABORATIVE DESIGN IDEATION, CURATION, LEARNING AND EVALUATION IN CREATIVE CONTEXTS

An Undergraduate Research Scholars Thesis

by

AARON PERRINE

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Andruid Kerne

May 2019

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

LiveMache: Supporting Collaborative Design Ideation, Curation, Learning and Evaluation in Creative Contexts

Aaron Perrine
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Andruid Kerne
Department of Computer Science
Texas A&M University

This project investigates how we can computationally support phases of the design process in solving creative problems, as well as the methods that arise in its educational context. Prior work addresses foundational topics, such as free-form web curation and creativity, which are necessary to ground the motivations in designing a system to provide this kind of context. Combining this background understanding and ongoing discussions with design instructors clarifies what a tool must include to viably support individuals involved in creative processes. What arises is a need for a system that implements real-time collaboration, as design projects are often collaborative, whose space and functionality does not limit a designer's creativity.

Our solution is LiveMache, a web application that provides live, collaborative capabilities for collecting and organizing content, along with writing sketching, chat, and live streaming video. Although the user interface design is just as important in creating this application, my research particularly focuses on how to make the user experience as expected, which depends on the database, the client/server architecture, and role-based access control. This paper addresses the reasoning behind the design of these three components for LiveMache, and discusses how their functions serve to make LiveMache successful as a creativity tool.

CHAPTER I

INTRODUCTION

This research investigates the development of collaborative design curation spaces as transformative cyberlearning support for design teams solving creative problems. Design team projects have been shown to improve student retention, satisfaction, and learning performance in engineering education (Dym et al., 2013). Unfortunately, design is difficult, and novice designers face creativity challenges such as fixation, depth-first thinking, and an unwillingness to abandon their original concepts. Conjointly, when students collaborate on teams, social and organizational challenges compound (Rozovsky, 2015). These problems become magnified by issues of communication, coordination, and contextualization during remote collaboration online (Hilliges et al., 2007). To address these problems, this research focuses on the development of a multi-scale designcuration space application called LiveMache that supports team collaborative design curation processes.

We draw on culture and creativity to frame these activities through an art tinted lens of curation, which curatorial scholar Paul O'Neill has called a process that art practitioners perform involving the bringing together of elements to conceptualize and create a new context for purposes of reflection and communication; through curation, works are found, collected, interpreted, and visually arranged, in an exhibition space to stimulate active engagement (2012).

Our research uses a design curation method, which emphasizes the use of scale and space to organize and relate artifacts. This method seeks to use varying levels of scale and an infinite space canvas to capitalize on human spatial reasoning, and to facilitate understanding, communication, ideation, and creativity (Hornbæk, Bederson, & Plaisant, 2002). This extends itself to collaborative multiscale design curation involving groups of people working together to visually and conceptually organize their artifacts in this zoomable space while using panning and zooming as principal interface interaction techniques.

CHAPTER II

PRIOR WORK

We discuss related work on web curation and how it applies to supporting users in creative contexts. Web curation has become a popular social media activity, in which people assemble boards and engage in everyday ideation (Linder, 2014), around a variety of topics including designs, recipes, photography, tutorials and much more (Zarro & Hall, 2012). To expand this concept, we turn to art, where *curation* is the creative conceptualization and design of a context, in order to develop new ideas (O'neill, 2012). Works are arranged and interpreted in an exhibition space, to stimulate active engagement and produce meanings significant to participants. Curation serves as a means for framing and conceptualizing how creative work and its contexts are understood. Central to curation is the art practice of *assemblage*, the way of making a creative work by fastening found object materials together (Seitz, 1961). Assemblage showcases the duality and tension between the original and resulting contexts. Curation is a form of the time-tested practice of `doing research in the library' (Stoan, 1984). We articulate a definition of *free-form thinking*, as a creative cognitive process of open exploration, association, improvisation, synthesis, emergence, and ideation (Linder et al., 2015).

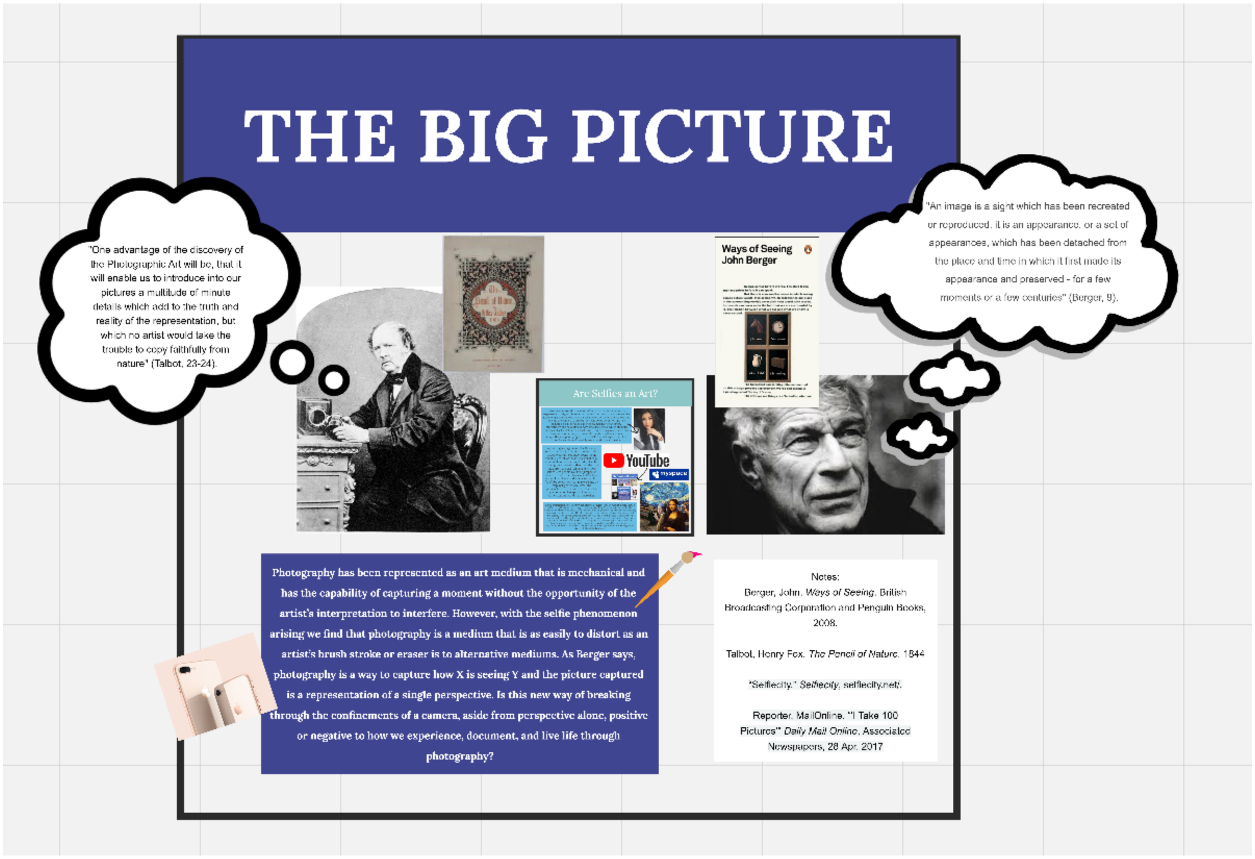
Building on these ideas of curation and free-form, we extend our prior work on *information composition* (Kerne et al., 2014). We invented *free-form web curation* to support users in creating new conceptual, spatial contexts, in which they discover and interpret relationships through visual thinking, while composing content elements to form a connected whole. *Free-form web curation* integrates collecting web content elements—including text, images, maps, and shared documents—and sketching, with assemblage, in a space. A zoomable user interface (Bederson, Meyer, & Good, 2000) gives users continuous traversal of curation spaces, with changes of scale, as a principal means of organizing content and navigation. *Free-form web curation* extends spatial hypertext, advancing its ability to help users avoid premature formalism, that is, representations which demand structure before a participant is ready to articulate it (Shipman & Marshall, 1999).

Supporting creative context requires building an understanding of creativity. Creativity has been approached from social, learning, and cognitive perspectives. An early definition, by the social psychologist, Amabile, defines creativity as that which participants in a situation agree is creative (1983). Subsequently, she defined creativity in organizations as, "...the production of novel and useful ideas in any domain," which are different than what has been done before (1996). She says that creativity is a social process involving expertise, creative-thinking skills, such as flexibility and imagination, and motivation which must be fostered, through challenges, freedom, resources, and encouragement (1998). Kaufman and Beghetto (2009) investigate creativity in learning. They conclude that much of student design team work lies within the creativity spectrum. Creative cognition researchers identify multiple types of cognitive processes as creative, such as ideation, insight, conceptual combination, analogical reasoning, restructuring, visual synthesis, and visualization (Finke, Ward, & Smith, 1992). In addition we have conducted prior work on collaborative design difficulties, including how we can develop techniques for supporting shared awareness (Brown, 1992) and the modulation of individual and collaborative work to help prevent team design fixation (Nicol & Macfarlane-Dick, 2005).

CHAPTER III

METHODOLOGY

To develop and study LiveMache, we investigate current technologies and work with students and instructors across a range of university design and team project oriented courses. We take a participatory approach, working with instructors and students as co-designers rather than as subjects (Lupfer, 2018, p. 13). We have worked with instructors throughout several disciplines, including english, architecture, and engineering, to develop LiveMache specific assignments, such as the example shown in Figures 1 and 2, which integrate with instructors' existing design curricula. LiveMache assignments focus on the ideation, curation, and evaluation stages of the design process. For data collection and analysis, we take a mixed-methods approach, involving qualitative and quantitative methods. Quantitative data include user event logs. Qualitative data include interviews with students, instructors, and researchers, along with feedback on the application provided by LiveMache users. We have evaluated users' curation artifacts and their associated operation logs, analyzing this data both quantitatively and qualitatively by combining the following evaluation methodologies: visual grounded theory (Lupfer, 2018, p. 10). and curation metrics (Lupfer, 2018, p. 7). In keeping with our participatory approach, we work closely with instructors to better understand the students' work and learning outcomes, allowing us to continue iterating on LiveMache's design. While there is research to be found on the technologies and design methods chosen, such as evaluation studies on Node.js and websockets (Chaniotis, Kyriakou, & Tselikas, 2015), many of the design choices throughout the development of LiveMache were based on best practice community driven guides for the open source tools chosen.



THE BIG PICTURE

"One advantage of the discovery of the Photographic Art will be, that it will enable us to introduce into our pictures a multitude of minute details which add to the truth and reality of the representation, but which no artist would take the trouble to copy faithfully from nature" (Talbot, 23-24).

"An image is a sight which has been recreated or reproduced. It is an appearance, or a set of appearances, which has been detached from the place and time in which it first made its appearance and preserved - for a few moments or a few centuries" (Berger, 9).

Photography has been represented as an art medium that is mechanical and has the capability of capturing a moment without the opportunity of the artist's interpretation to interfere. However, with the selfie phenomenon arising we find that photography is a medium that is as easily to distort as an artist's brush stroke or eraser is to alternative mediums. As Berger says, photography is a way to capture how X is seeing Y and the picture captured is a representation of a single perspective. Is this new way of breaking through the confinements of a camera, aside from perspective alone, positive or negative to how we experience, document, and live life through photography?

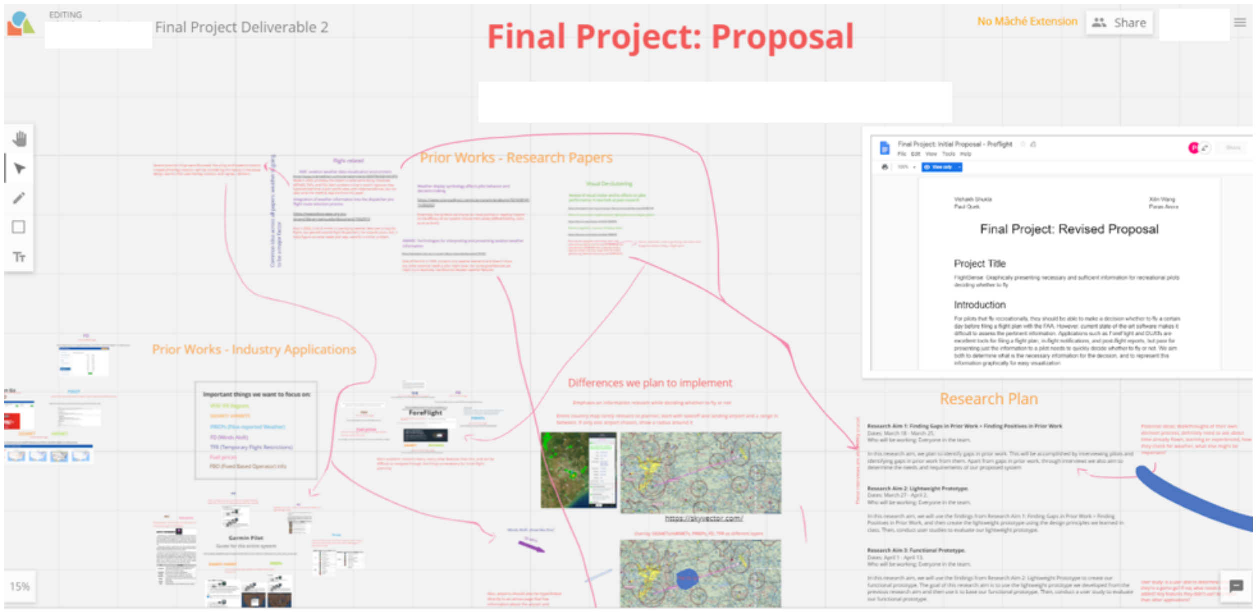
Notes:
Berger, John. *Ways of Seeing*. British Broadcasting Corporation and Penguin Books, 2008.

Talbot, Henry Fox. *The Pencil of Nature*. 1844

"Selfiecity." *Selfiecity*, selfiecity.net/.

Reporter, MailOnline. "I Take 100 Pictures" *Daily Mail Online*, Associated Newspapers, 28 Apr. 2017

Figure 1. The Big Picture. This is an example of a LiveMache design curation.



EDITING
Final Project Deliverable 2
Final Project: Proposal
No Mâché Extension
Share
Prior Works - Research Papers
Prior Works - Industry Applications
Differences we plan to implement
Research Plan
ForeFlight
Garmin Pilot
https://skyvector.com/
15%

Final Project: Revised Proposal

Project Title

Introduction

Figure 2. Final Project Proposal.

CHAPTER IV

DATABASE DESIGN AND DATA MODELING

The design curation elements and records, which users create using LiveMache, are stored in different databases depending on their data type. With over 49 million user events, and roughly half a million user created records, proper database architecture is vital for creating a responsive and robust application, which is capable of supporting participants engaged in real-world design activities. Analyzing and filtering the vast amount data that LiveMache provides and providing a seamless experience for the user call for a proper data modeling design within each database. Together, the databases formulate the backbone of the architecture, and their design becomes vital for developing a cohesive application.

The four primary databases that support LiveMache store the following data types: application objects, application logging, user event logging, and session state / socket objects. Below we present each database's primary purpose, and the supporting design.

Application Objects Database

The application objects database supports LiveMache's real-time functionality, storing objects—such as user account information, design curation sub-objects, and user groups—created and modified during the application's use. This database contains all documents instantiated from LiveMache's data models, the most important being *users* and *curations*. From the user and curation models, we create the organizational models, *groups* and *folders*, where a group may contain many users and many folders, and a folder is a collection of many curations. There are other objects that are stored in the application objects database — such as images, roles, and elements — though articulating their purposes and relations between each other would be extensive. Figure 3 is a simplified and reduced mongo schema layout, showing the core objects, as well as their relations and attributes. Nonetheless, the well-defined relations between all objects create the database's structure, which dictates the application logic.

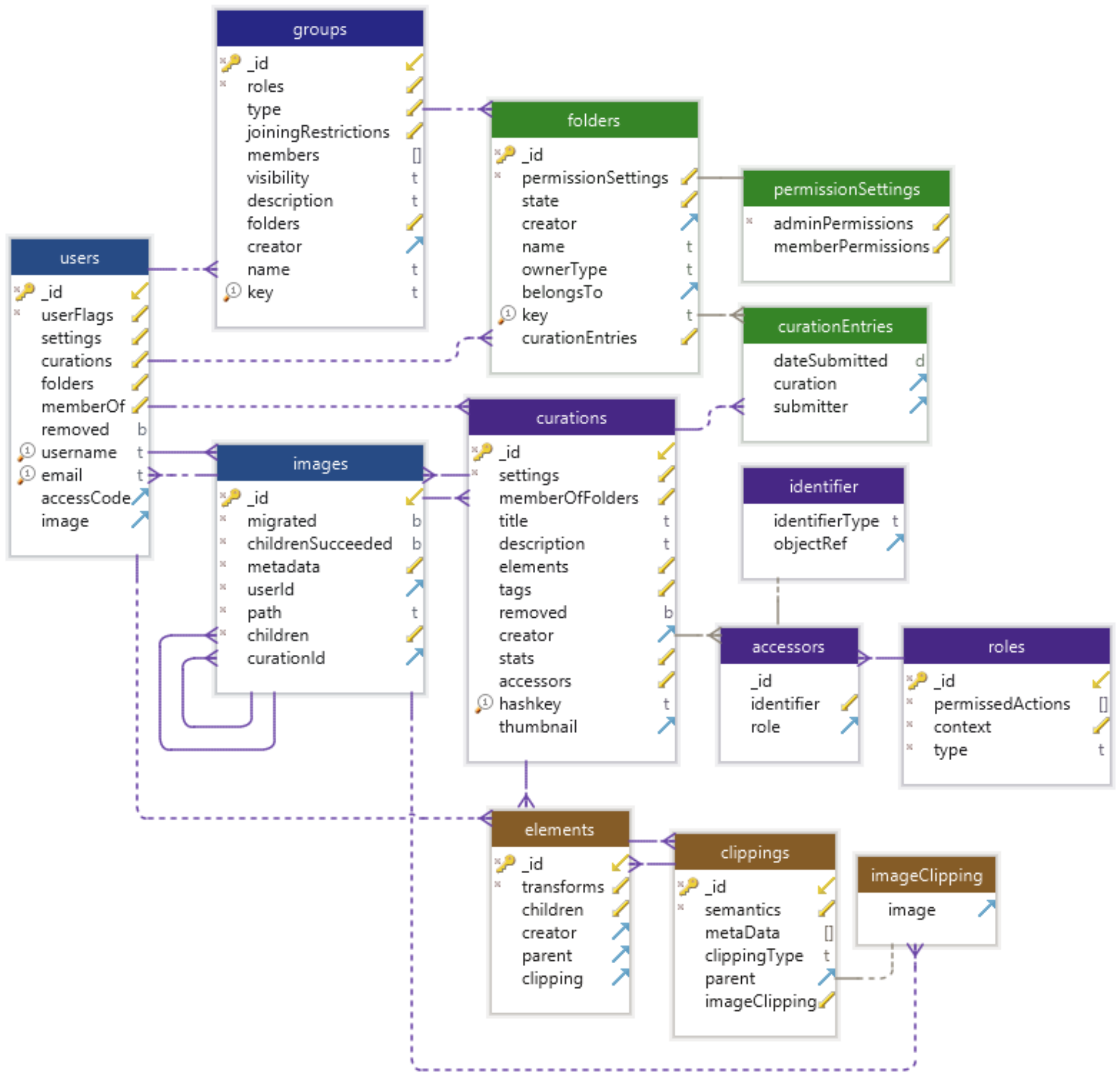


Figure 3. LiveMache Schema. This is a simplified and reduced schema layout of the application objects database

Middleware at the schema level plays a vital role in LiveMache document interaction and runs directly before or after making a query to the database, allowing for further control on the data entering and leaving. It functions as a process helper and ensures that attributes, such as a user's salt or oauth metadata, are never returned outside of authentication specific queries.

While the logging database is intended to grow quite large, objects in this database are frequently requested, modified, and returned. When users are collaborating on a curation, the curation or one of its elements can undergo this modify and request process over 100 times a second. As a curation containing all of its elements and clippings can grow quite large, this data transmission cycle can slow down the frame rate of a client's view, preventing a real-time collaborative experience. The two primary solutions to this problem were (1) efficient data storage and (2) only returning the minimal amount of data needed. (1) is solved by only storing the most recent document changes and isolating binary data to a file system. While previous edits are important for analytic purposes, the event logs are comprehensive enough to generate a curation history. (2) is solved with the implementation of a query depth field that is passed with the query and handled by the middleware. This allows the return of only the data needed by the client in order to update the user’s view, oftentimes decreasing the data transmission costs by a magnitude of 100.

Another primary design component both for conducting analysis and application maintainability is a set of well-defined rules and fields that each schema must follow. Given the versatility of data that a user can import into a curation, some fields should be flexible and support multiple data types. Each of these data types must then pass a series of validation, categorization, and manipulation processes to properly label and store the data in the application objects database, allowing for speed, security, and future analysis.

Application Logging Database

The application logging database stores relevant information regarding the application’s events, which are used for debugging and system analysis. From server initialization to the handling of each request, there is some type of potential inner application log that can be made. These log events, which include new connections, server errors, and unexpected data types, are then categorized and stored. As the user does not ever interact with this database, speed of retrieval and storage constraints do not motivate the application logging database design. With the most common use being error analysis, the most important design component was the proper collection and labeling of logs. To accomplish this, there are established logging levels describing the type of the event coupled with copious amounts of context data. Because of this type of data logging can grow quite large in size, the database is exported daily to an external harddrive and soon after, flushed.

User Event Logging Database

Every user action in LiveMache is stored as an event in the user event logging database. These logged events are essential in providing data for investigating and understanding how LiveMache users collaboratively approach creative problems. Designing a curation emits a multitude of events, often many per second, these are stored as a json-blob containing metadata about the event and a reference to the user, and curation. These events include the following: element position changes, zoom scale, other user interaction, element creation, interface interaction, media streaming, etc. Thus far we have collected over 49 million events and as each event contains the metadata and objects being operated upon, the size of this database has grown quite large. As the user has no need to query this data, storage and retrieval constraints are of little worry, however, as this data is vital for analytics, it is important for developer sanity to implement logic that can handle this data efficiently. Most often this has taken the form of curried helper functions that can be partially applied and reused several times. This allows us to run several sets of analytics on the data in a single round, often times reducing the length of execution by several fold.

Session/Sockets Database

The session/sockets database primarily stores data related to a users session or active curation, which acts as a cache for all of the collaborating users. When a user logs into LiveMache, a user is assigned a session object, which is passed between the client and server for housekeeping purposes, such as maintaining the user's authorization and state. Because these kinds of requests are constantly running, the database must be extremely fast, and the stored data should be minimized in size. These requirements in speed and size are even more necessary when considering real-time collaborative use, as users within a curation can see each other's actions in real-time. For example, all users should be able to see a user moving an image while it is occurring. Because we are aiming to keep the users frame rate between 60 and 30 frames per second, this action would potentially require 60 updates a second within the application database. However, by using the sockets database as a cache we are able to broadcast this movement event to other users in the curation then update the modified curation within the application database only at the end of specific modifying actions. Consequently, the design for this database implements a Redis database, specifically created for the types of tasks explained above. Redis uses a key-value storing method that can handle hundreds of thousands of requests each second and specializes in transferring small data, making it ideal for supporting LiveMache collaboration.

CHAPTER V

CLIENT SERVER ARCHITECTURE

LiveMache is an application that must be able to support thousands of users performing computationally intensive tasks, such as manipulating images and managing media streams. Our goal for each curation is to support upwards of 30 users collaborating on these tasks in real time.

To accomplish this, the LiveMache system is multi-paradigmatic. There are traditional HTTP REST APIs, for managing users, groups, and curation objects, and a WebSocket architecture for handling real-time, collaborative curation activities. Both of these paradigms are built using Node.js, an asynchronous, event-driven Javascript runtime, designed to build scalable web applications. However, beyond this, the handling of clients differs greatly in design and will be discussed separately.

Server/Client Object Management

Within LiveMache there are several possible object request and management tasks that user may choose to perform, such as updating account information, managing / retrieving curations, interacting with their groups, etc. All of these tasks are handled by a series of HTTP requests that are made to the server. This process can be reduced to the request made by the client, and the response back from the server.

Server Request Handling

When a request is made from the client it must pass through several different sets of request pre-processors called "middleware." The middleware handles steps such as authorization, request validation, session management, etc., such that by the time the request arrives at the handler, it is operable. All routing logic supports the following five processes: collection, validation, filtering, data modification, and response handling. To illustrate this, consider a user making a request to edit a group's folder title. The process would involve the following steps:

1. Collection: A curation query is generated using the locators (discussed in the next section) and sent to the database to retrieve all the documents needed.
2. Validation: The client and the data being requested are checked using the later described RBAC validators, along with additional route specific logic. In this case, the logic center around determining whether the user has access to make changes to a folder within the group.
3. Filtering: The retrieved data is filtered based on the additional permission qualifiers, e.g., if this is a group for a course, the instructor may not allow students to view folders that contain curations owned by other classmates. Thus, these folders would be filtered out and hidden from other students.
4. Data Manipulation: This is route specific logic that performs the intended action of the request, in this case an update query would be formed with the user and folder
5. Response Handling: This is the requested / updated data or the client error message if something went wrong.
6. Logging: Throughout all route handling, there are several points where some form of logging, such as errors, warnings, data modifications, etc., may occur. These logs are handled by the logging service and not sent back to the client.

Client Request / Response Handling

The client side of LiveMache requires fine grained functionality and employs many more request functions than the server exposes. In order to support the required specifics of these tasks, the client must be able to compose several of the exposed server side routing results into the desired output. However, the servers' exposed routes cannot be too generalized or request friendly less they sacrifice security. Thus there are difficult design choices when deciding how the client side of the application communicates with the server.

While LiveMache must support thousands of users, it is still considered a fairly small web application by modern standards. This allowed us to relax the emphasis on generalization. However, some amount of flexibility, such as dynamic queries, is needed. This is accomplished with a construct we call "locators," which is a query type object that must follow a set rules, increasing query flexibility, but only as needed. When a client makes a request to the server, the client begins by constructing these locators and passing them with the request, which are then used to generate the true query on the server side. Continuing with the previous example, depending on the context of where and how the client is requesting curations, other options such as retrieval depths may be passed with the locator to determine how much of the database document should be retrieved.

Real-time Web Sockets

When users are in the collaborative curation portion of the application, they must be able to make real-time edits and view the edits made by others. This requires an event driven architecture, with a focus on speed and atomic data manipulation. To accomplish this, our system utilizes the Web Socket protocol. While web sockets are similar to HTTP, in that they are both TCP-based, the primary difference is the fast bi-directional communication, in which web sockets excel. We can utilize this speed to enable many real-time collaborative curation sessions via a framework called "socket.io."

Design Considerations

Needing to support thousands of users, this application cannot run on a single machine. Furthermore, because Node.js is single threaded, each machine that runs our system is split into a cluster of Node.js processes. This creates an interesting design problem that arises in the case of multiple users interacting with the same element within a curation, which can be potentially split between different Node.js processes. Therefore, the state of a curation space must be synced with all users. We use the Redis datastore to sync the socket in a fast-access database, letting Redis act as a glue between the distributed services.

As users collaborate, another problem that arises is when one user tries modifying a curation element concurrently with another user. To solve this, a concept of element locks are introduced. In the case of conflicting actions, the first modifying user possesses the lock and other users are unable to modify the element until the conflicting action is stopped, thereby releasing the lock.

Socket Event Cycle

Throughout a users' design session, there are three guaranteed event processes: connection, curation event loop, and disconnection.

Connection

When a client connects to a specific curation space, a socket is created between the client and the server and places this client in a room specific to the curation. During the connection initialization process, a series of construction functions use the socket to initialize event routes between the client and the server. This enables faster event handling and a better logging flow with each connected user

Curation Event Loop

As soon as the initialization process is complete, the user enters the main event cycle. During this time are many events per second that can be divided into handled events and strictly logged events. As the name suggests, strictly logged events are events that do not change the state of the curation, but are perhaps interesting for future analysis, and thus sent to the logging database. State changing events, such as the movement or rotation of an element, the importation of media, the sending of a chat message, playing/pausing a stream, etc. These are then emitted via the initialization of the socket, which the server then parses and responds back.

When the server receives an event, it first routes the event to its specific handler, which then performs up to three actions: an atomic data manipulation, an emit back to the client, and a broadcast to all clients in the room. The atomic manipulation is necessary to guarantee that parallel writes do not corrupt the integrity of the data. Most commonly a user will perform some curation state change, which will be broadcast to all other users in the room. Immediately after receiving the broadcast, the client side will update the necessary part of the view for the connected user.

Client Side Optimization

A large amount of effort has been put into maintaining a design environment that obstructs the user as little as possible. This requires that the manipulation of design elements and importation of media appear seamless. While several optimization techniques have been implemented, the primary bottleneck is media importation. The browser can efficiently handle the majority of the caching processes, but when many users are collaboratively working on a curation with hundreds of images, performance issues and crashed clients become a definite concern. To combat this problem, we implemented a scale based optimization technique (Bederson & Hollan, 1994; Perlin & Fox, 1993) that uses the users' viewport to determine how far we can reduce an object to increase performance. For example, when there is a large amount of images in a curation and the user's view is at a low zoom level, there is often a subset of images whose resolution can be lowered without disrupting the user's experience, let alone being noticed at all. These images have pre-created copies called "siblings," which are of fractions of the original resolution and used to replace images within this subset on the fly. This allows us to save much of the memory that large zoomed out images would otherwise consume.

CHAPTER VI

ROLE-BASED ACCESS CONTROLS

Within LiveMache a user can perform many different actions on objects created by themselves or by other users. At a high level, these objects are the curations, users, folders, and groups. The concept of these objects evoke many design questions regarding what actions a user can perform on a specific object in a specific context. In an example of an education context, an instructor using LiveMache may create two folders, one containing examples of curation submissions and one for the students' actual submissions. In this case, all students may be permitted to view all curations in the example folder, but not all curations in the submission folder. More generally, owners of a group should have the functionality to control curations and folders' access permissions of other members in the group. The example becomes more complex when a group owners wants to further customize access control on particular subsets of a group's members, restricting them to selected actions and curations within the group's folder.

Contextualizing this example, we present a situation that occurred within one of the courses using LiveMache. A design instructor created a group, inviting the students and teaching assistants as members. The teaching assistants needed to be able to comment on all curations within the group's folders. While the students needed to be able to view all curations within the groups folders, they only needed to modify their own curations, or those of their teammates. Thus, creating a system that can incorporate these varying levels of access and types of roles is important for creating an environment used to support students solving creating problems in their learning processes.

Designing for flexible access controls

Because of the variable user types, actions performed, objects, and contexts, developing for case by case conditions is insufficient. This is a common problem within large scale user-based systems, and the common solution is the implementation of role-based access controls (Ferraiolo, 1995; Sandhu, Coyne, Feinstein, & Youman, 1996). Generally, RBAC enables the creation of role objects, associated with tasks that require permission to perform them. For LiveMache, user objects are assigned a role object, acquiring the role's permissions. A full RBAC implementation includes role combination and role hierarchies, with higher roles subsuming sub-role permissions. This more complex implementation is most suitable for operations conducted by millions of users of large scale systems, such as Microsoft's Azure Active Directory suite, which requires millions of different types of users and user-owned objects. As LiveMache is not intended to support this magnitude of users and roles, a full RBAC implementation is unnecessary, and ultimately, not worth the effort in development.

RBAC in LiveMache

Because LiveMache is a system created around a few core objects (curations, groups, and folders), it only needs to incorporate a reduced version of RBAC, while still remaining flexible enough to support current and future research goals. Roles in LiveMache are distinct database objects, which require a context, a type, and a set of permitted actions. The context of a role helps determine how the role should be handled. Currently, the three contexts a role may be associated with are user, group, and folder. The type of a role is a human readable name, such as "owner," from which the set of permitted actions is sensible. Each of the core objects relate and utilize the roles in a different way.

Curations

Curations and their sub-objects are the objects most frequently operated on, thus containing a large amount of the RBAC logic. Each curation contains a list of accessors, which is an object binding the object accessing it, and the associated role. For example, an owner of a curation in LiveMache is given a role object of context “curation”, type “owner” and a list of permitted actions they are allowed to perform on this specific curation, such as view, edit, comment, etc.

Curation Folders and Groups

While folders and groups are motivated by organizational purposes, they also allow for the creation of different permission settings, which are then appropriately propagated to the curations or members that they contain. For example, if User A owns Folder B, which exists in Group C, and User A wishes to permit viewing access to other group members, each curation within Folder B is given an accessor, which is created in relation to Folder B, Group C, and the corresponding curation. If another user in Group C, we will call him User D, then tries to view one of the curations in Folder B, User D’s accessors are queried and reduced, such that the accessors inform LiveMache that User D is a member of Group C. The final step checks User D’s actions against the permitted actions of the role he is assigned.

Security and Error Handling

LiveMache contains large amounts of user confidential data, which may come from user account information, event logs, or chat messages. Furthermore, because LiveMache is an asynchronous, event driven application, errors can be difficult to handle, especially when thrown out of their originating context. Consequently, security becomes one of the driving factors in LiveMache’s development. To appropriately deter attacks, we have implemented and followed a wide range of the best security practices specific to the technologies used. Docker, a containerization tool, is used to wrap LiveMache's production environment in an isolated set of containers. Docker runs on a virtual machine, which is located within our main server, implementing many of the front-facing security measures. Although Docker plays a significant role in LiveMache's security, a full discussion on the server is out of scope. Thus, we continue focusing on LiveMache and expand on its neighboring containers that exist within the same virtual machine, explaining the core aspects of the design.

Neighboring Containers

There are four containers within the LiveMache virtual machine: Nginx, Redis, Mongo, and Node.js. All are well established software tools that have many of their best practices baked into the image from which the container is instantiated. Thus, much of the work for the Redis and Mongo containers reduces to proper network configurations, environment variable injection, and database authentication configuration.

Nginx

A Nginx server acts as a reverse proxy and load balancer and is important for routing between a cluster of Node.js processes. There are several additional security measures Nginx provides, such as forcing TLS/end-to-end encryption, hiding certain proxy headers, manipulating response headers, etc. However, as the front-facing server implements many of these measures, the virtual machine hosting LiveMache uses the Nginx container primarily for sticky sessions between Node.js instances and load balancing.

Application Security / Error Handling

If several users are collaborating within a single curation, an error caused by one user should not disrupt another user. Also, a true server error, such as an accidental release of data to non-authorized users, is a concern that cannot be taken lightly. Although general security concepts—e.g., authentication, oauth handling, password salting, and general encryption processes—are extremely important, their standard implementation does not require further discussion.

A primary solution to the error described above, is (1) to ensure an error is always caught and (2) to utilize middleware. We accomplish (1) by creating a multiple level catch system that starts at a route and propagates up to the highest scoped catch. This design allows an error, such as permission validation, to be caught and handled within the route it is specifically in, while still catching decontextualized errors that may occur outside the originating process. Decontextualized errors are unexpected, and therefore, treated more harshly. If an error hits the top-level catch, no data, aside from an error message, is returned to the client. We heavily rely on routing middleware and database middleware to accomplish (2). The security implemented in the routing middleware handles a request, passing the request to more specific handlers. When any request reaches LiveMache, the request first passes through general HTTP sanitation and parsing, then reaches the more specific such as the authorization middleware. As discussed earlier in the database design, database middleware allows for fine tuning control on what enters and leaves the database, serving as a last check for validation and modification of a query. For example, when a user is requested from any context outside of authentication, their authentication data will not be returned or if a user tries to spoof the query builder and request data that should remain in the database for analysis, it will not be returned.

CHAPTER VII

CONCLUSION

This research has investigated the design of a system to support collaborative design ideation, curation, learning, and evaluation in creative contexts. We have implemented a multi-scale, real-time collaborative curation space called LiveMache that seeks to answer this research question. LiveMache is currently being used in a variety of courses and has over 400 active users. The application continues to undergo development and should now be able to support thousands of concurrent users. By the end of this year we are planning a public beta release in order to reach more users, and study new contexts.

REFERENCES

Arnold, K. E., & Pistilli, M. D. (2012, April). Course signals at Purdue: Using learning analytics to increase student success. In *Proceedings of the 2nd international conference on learning analytics and knowledge* (pp. 267-270). ACM.

Amabile, T. M. (1983). The social psychology of creativity: A componential conceptualization. *Journal of Personality and Social Psychology*, *45*(2), 357-376. http://dx.doi.org/10.1037/0022-3514.45.2.357

Amabile, Teresa M. (1996, January). "Creativity and Innovation in Organizations." *Harvard Business School Background Note*, 396-239.

Amabile, T. M. (1998). *How to kill creativity* (Vol. 87). Boston, MA: Harvard Business School Publishing.

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *The journal of the learning sciences*, *13*(1), 1-14.

Bederson, B. B., & Hollan, J. D. (1994, November). Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology* (pp. 17-26). ACM.

Bederson, B. B., Meyer, J., & Good, L. (2003). Jazz: an extensible zoomable user interface graphics toolkit in Java. In *The Craft of Information Visualization* (pp. 95-104). Morgan Kaufmann.

Bradford, P., Porciello, M., Balkon, N., & Backus, D. (2007). The Blackboard learning system: The be all and end all in educational instruction?. *Journal of Educational Technology Systems*, *35*(3), 301-314.

Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The journal of the learning sciences*, *2*(2), 141-178.

Chaniotis, I. K., Kyriakou, K. I. D., & Tselikas, N. D. (2015). Is Node. js a viable option for building modern web applications? A performance evaluation study. *Computing*, *97*(10), 1023-1044.

Cugini, J., Kuhn, R., & Ferraiolo, D. (1995). Role-based access control: Features and motivations. In *Proceedings of the Annual Computer Security Applications Conference, Los Alamitos, Calif., 1995*.

Dennen, V. P., Aubteen Darabi, A., & Smith, L. J. (2007). Instructor–learner interaction in online courses: The relative perceived importance of particular instructor actions on performance and satisfaction. *Distance education*, *28*(1), 65-79.

Duval, E. (2011). Attention please!: learning analytics for visualization and recommendation. *LAK*, *11*, 9-17.

Dym, C. L., Agogino, A. M., Eris, O., Frey, D. D., & Leifer, L. J. (2005). Engineering design thinking, teaching, and learning. *Journal of engineering education*, *94*(1), 103-120.

Finke, R. A., Ward, T. B., & Smith, S. M. (1992). Creative cognition: Theory, research, and applications.

Hamilton, W. A., Lupfer, N., Botello, N., Tesch, T., Stacy, A., Merrill, J., Williford, B., Bentley, F., & Kerne, A. (2018, April). Collaborative Live Media Curation: Shared Context for Participation in Online Learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (p. 555). ACM.

Hilliges, O., Terrenghi, L., Boring, S., Kim, D., Richter, H., & Butz, A. (2007). Designing for collaborative creative problem solving. *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, 137-146. doi: 10.1145/1254960.1254980.

Hornbæk, K., Bederson, B. B., & Plaisant, C. (2002). Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *9*(4), 362-389.

Jain, A. (2017, June). Measuring Creativity: Multi-Scale Visual and Conceptual Design Analysis. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition* (pp. 490-495). ACM.

Kaufman, J. C., & Beghetto, R. A. (2009). Beyond big and little: The four c model of creativity. *Review of general psychology*, *13*(1), 1-12.

Kerne, A., Webb, A. M., Smith, S. M., Linder, R., Lupfer, N., Qu, Y., Moeller, J., & Damaraju, S. (2014). Using metrics of curation to evaluate information-based ideation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *21*(3), 14.

Kuznetsova, P., Chen, J., & Choi, Y. (2013). Understanding and quantifying creativity in lexical composition. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1246-1258).

Laflen, A., & Smith, M. (2017). Responding to student writing online: Tracking student interactions with instructor feedback in a Learning Management System. *Assessing Writing*, *31*, 39-52.

Linder, R., Snodgrass, C., & Kerne, A. (2014, April). Everyday ideation: all of my ideas are on pinterest. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (pp. 2411-2420). ACM.

Linder, R., Lupfer, N., Kerne, A., Webb, A. M., Hill, C., Qu, Y., Keith, K., Carrasco, M., & Kellogg, E. (2015, June). Beyond slideware: How a free-form presentation medium stimulates free-form thinking in the classroom. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition* (pp. 285-294). ACM.

Lupfer, N. (2018, May). Multiscale Curation: Supporting Collaborative Design and Ideation. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 351-354). ACM.

Nicol, D. J., & Macfarlane- Dick, D. (2006). Formative assessment and self- regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, *31*(2), 199-218.

O'neill, P. (2012). *The Culture of Curating and the Curating of Culture (s)*. Mit Press.

Perlin, K., & Fox, D. (1993, September). Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (pp. 57-64). ACM.

Rozovsky, J. (2015). The five keys to a successful Google team. Retrieved from https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (February 1996). Role-Based Access Control Models. *IEEE Computer, 29*(2), 38-47. doi: 10.1109/2.485845.

Seitz, W. C., Museum of modern art (New York, NY)., Dallas Museum of Contemporary Arts (Tex.)., & San Francisco Museum of Modern Art (Calif.). (1961). *The art of assemblage* (Vol. 19, No. 1). New York: Museum of Modern Art.

Shipman, F. M. & Marshall, C. M. (October 1999). Formality considered harmful: experiences, emerging themes, and directions on the use of formal representations in interactive systems. *Computer Science Cooperative Work, 8*(4), 333-352. doi: 10.1023/A:1008716330212.

Siangliulue, P., Chan, J., Dow, S. P., & Gajos, K. Z. (2016, October). IdeaHound: improving large-scale collaborative ideation with crowd-powered real-time semantic modeling. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (pp. 609-624). ACM.

Siangliulue, P., Chan, J., Gajos, K. Z., & Dow, S. P. (2015, June). Providing timely examples improves the quantity and quality of generated ideas. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition* (pp. 83-92). ACM.

Stoan, S. K. (1984). Research and library skills: An analysis and interpretation. *College & research libraries, 45*(2), 99-109.

Van Leeuwen, A., Janssen, J., Erkens, G., & Brekelmans, M. (2014). Supporting teachers in guiding collaborating students: Effects of learning analytics in CSCL. *Computers & Education*, *79*, 28-39.

Zarro, M., & Hall, C. (2012, June). Pinterest: Social collecting for# linking# using# sharing. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries* (pp. 417-418). ACM.