ShowNTell: An easy-to-use tool for answering students' questions with voice-over recording

Anand Bhojan[‡], Kwan Yong Kang Nicholas[‡], Nidhi Sharma[†]
[‡]School of Computing and [†]School of Science, National University of Singapore
Email: [‡]*banand,nichkwan*@comp.nus.edu.sg, [†]*phyns*@nus.edu.sg

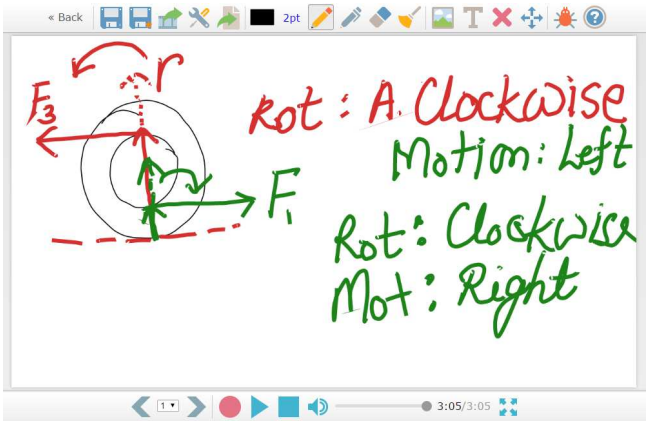

Fig. 1. ShowNTell Editor

***Abstract*—The use of recording applications for teaching is increasingly popular, with tools such as Ink2Go and ShowMe being available on many platforms and at reasonable cost. However, most recording applications available are typically native applications and do not work within the web browser. In this work, we study the feasibility of implementing a recording solution in the web browser environment.**

I. Introduction & Motivation

ShowNTell is an implementation of a screen recording application that runs within a web browser on a mobile device [1]. The application will provide an interface similar to a virtual whiteboard, and allows the recording and playback of drawings made on the whiteboard (see Figure 1). Audio recordings are supported to facilitate the creating of lessons and tutorials within the application for the purposes of eLearning.

We describe our motivations for this work in the following sections:

A. Web platforms

New browser technologies under the umbrella group of HTML5 technologies have provided new opportunities for new types of applications to run within the web browser, providing services normally done by native applications. For instance, the introduction of the 2D Canvas API allows images to be generated in real-time within the web browser, instead of being generated on the server [2]. This paves the way for applications such as interactive games and media that requires graphical content to be updated in response to user input.

Improvements in communication technologies such as access to higher and more stable bandwidth as well as faster transfer speeds can allow us to provide real-time services that may require continuous transfer of relatively sizeable amounts of data from the client device. This is crucial for ensuring responsiveness of the application which has data dependencies on content on the server.

As many mobile devices have access to the Internet, we would also perform user experience studies on the use of touch-based interfaces for the use of recording and drawing, which would be one of the primary input interfaces for ShowNTell. Many mobile browsers also support the new APIs that are available on desktop browsers, allowing our application to potentially work on those platforms without the user requiring installation of an application.

As a result of these emerging changes, it is now possible to implement a recording solution that runs on a web browser. Thus this acts as a primary motivation for pursuing this project. Our project would attempt to implement an application with native-like capabilities on the web platform by using APIs available on web browsers, as well as evaluate techniques that can potentially improve its usability or responsiveness in a web browser environment. Additionally, we would attempt to test our implementation on multiple platforms under different conditions, in particular the desktop and mobile browser environments.

B. eLearning

Recording applications are increasingly popular tools for the purposes of education and entertainment. For instance, the webcast system provided by the National University of Singapore (NUS) is able to generate automated recordings of a lecture in modules supporting that feature. Other modules have used other creative means aside from recordings such as gamification to improve the learning experience of the content in the module [3][4]. eLearning serves as an alternative or supplementary means for students to revise on topics that were

This work is supported in part by the National University of Singapore Learning Innovation Fund Technology (LIF-T) under the research grants C-252-000-113-001 and C-252-000-117-001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency, National University of Singapore.

covered earlier through traditional mediums. For our project, our main focus would be towards approaches that involve recordings.

Recordings are also used as tools to illustrate ideas that are typically difficult to describe as text or images [5]. This is because with the inclusion of time-based events, animations such as circling or highlighting become more prominent. Additionally, annotation mechanisms such as free-style drawing are easier to use than other traditional inputs for displaying mathematical equations or simple diagrams. This coupled with audio capture can allow for easier conveying of ideas that might be difficult to convey with a static document.

In ShowNTell, we attempt to build a recording tool that works in a web browser, allowing the application to work on a wide variety of platforms without requiring the installation of additional software. Unlike recordings used for lectures, ShowNTell is designed for shorter recordings that typically last for a few minutes, typically sufficient for answering questions or explaining concepts. The tool would act as a complement to the existing eLearning tools available.

II. Related Work

The idea of a whiteboard and video recording application is not a new concept, with many such applications available in the market. In this section we study several different types of applications that support screen recordings, enabling us to adopt similar techniques that allow us to manage data generated in recordings.

A. Desktop Implementations

There are many applications that support recording capabilities available on computers, such as CamStudio, Camtasia, and Open Broadcasting Software [6][7][8]. These applications are typically screen recorders which capture video input from the graphics hardware, usually through system APIs such as bitblt [9]. Screen recorders are applications that poll the screen and stores what is displayed into a file, thus capturing all visual details that are visible to the user. Screen capture applications may also capture audio from the audio input hardware. Other screen recording software may support annotation capabilities that allow adding details to the currently displayed screen, such as Ink2Go [10].

As recording generates cumulative data, different applications have different methods of handling the recording. Typically, a video recorder will continuously write frames to a file rather than retaining them in memory in order to keep memory footprint low. Ideal file formats for this include AVI files [11], as these files support the appending of video data into the file with minimal performance overhead. In other implementations such as Fraps [12], the recording may be distributed into multiple files, moving from one file to the next when a file size reaches a certain threshold . Distributing a recording into multiple files enables longer duration recordings on file systems that have an inherent file size limit, such as the 4GB limit in FAT32 systems [9].

In ShowNTell, we have adapted the file splitting approach which splits recording data into multiple fragments. This enables handling of certain data types such as audio much more easily, as well as allowing for responsive save times by allowing part of the document to be saved in the background while recording is ongoing. This approach is preferred over the single-file method because in a web browser environment, uploading of data to the server takes much more time compared to writing a file to the hard disk. Thus, we can reduce the perceived upload time by simply uploading parts of the earlier recording during the recording process.

B. Mobile Implementations

Mobile applications that support recording capabilities do not typically capture the screen, in contrast to desktop applications. Due to the sandboxed nature of mobile applications, capturing the screen is typically not allowed [13]. However, recording software with annotation capabilities and microphone support are widespread, such as ShowMe and ExplainEverything [14][15]. These fully utilize the touchscreen to provide an input suitable for free-style drawing.

Mobile applications face a larger resource constraint compared to desktop applications; in particular file-system access and computational resources are not a luxury. Thus recordings will typically record the input or the sequence of actions used to render the current scene, instead of recording the entire screen as a video. For example, when the user performs a free-style drawing, instead of capturing the image of the screen, it captures the coordinates needed to reproduce the same effect. This method of replaying events to simulate a recording will greatly reduce the storage, computational, and memory footprint of the recording, since only events will need to be stored instead of image frames. In the event a video output is desired, those events can be replayed to generate the video frames to produce the video.

In ShowNTell, we have attempted to emulate the same feature set provided by these applications, which include slide support, document import, and video export. Similar to ShowMe and ExplainEverything, ShowNTell will not actually record the screen contents, but instead the events to reproduce the visual state of the screen.

C. Web-based Implementations

Due to the resource requirements and limitations of existing browsers, there are not many recording applications available as a web application. There are some implementations such as BigBlueButton that provides recording and collaboration capabilities [16]. However, those implementations require the use of plugins such as Adobe Flash. Such plugins are typically not available on mobile devices due to either deprecation of the existing software packages or prevention of installation of the plugins by the operating system [17][18][19].

While there are web-based applications that provide drawing/annotation capabilities, or even provide the ability to collaborate with others over the internet, there are not many

TABLE I
API SUPPORT ON DIFFERENT BROWSERS.

	C	**FF**	**IE**	**S**	**C(A)**	**FF(A)**	**S(iOS)**
Version	*42*	*37*	*11*	*8*	*42*	*37*	*8.3*
Canvas	Y	Y	Y	Y	Y	Y	Y
IndexedDB	Y	Y	Y	Y	Y	Y	Y
UserMedia	Y	Y	N	N	Y	Y	N
Audio	Y	Y	N	Y	Y	Y	Y
Worker	Y	Y	Y	Y	Y	Y	Y
Fullscreen	Y	Y	Y	Y	Y	Y	N
FileReader	Y	Y	Y	Y	Y	Y	Y
XHR2	Y	Y	Y	Y	Y	Y	Y
Blob	Y	Y	Y	Y	Y	Y	Y

C = Chrome, FF = Firefox, IE = Internet Explorer, S = Safari

C(A) = Chrome (Android), FF(A) = Firefox (Android), S(iOS) = Safari (iOS)

complete solutions that support the ability to record the annotations and drawings for later playback. However, with newly available HTML5 APIs, it is now possible to implement a drawing and annotation web-based application with recording support.

Because HTML5 is in the emerging state, most browser vendors have only implemented a subset of it [20]. Nonetheless, some browsers such as Google Chrome have implemented most of the required APIs on publicly available versions of the browser, providing feasibility to the implementation and deployment of our application.

III. System Design

As ShowNTell is a web-based application, we have chosen to use a client-server model for the application. The client will be designed to run within a web browser, and does not require the installation of additional software aside from the web browser. The server will use a normal LAMP stack to serve web pages and store data, as well as NodeJS for the video rendering and document import systems.

A. Feasibility Analysis

Prior to and during the implementation of the project, we have evaluated whether the project can be implemented using available APIs and tools. Preliminary analysis has given the conclusion that majority of the browsers on desktop environments have support for the features that we require for our application, particularly Canvas 2D and Media Capture APIs [2][21].

On the mobile platform, Google Chrome and Firefox on Android offer the features that we require. Browsers such as Safari on iOS and Internet Explorer on Windows RT do not support the Media Capture API, thus features requiring the use of the microphone would be disabled on these platforms. Nonetheless, because a sizeable fraction of the browser market supports the APIs that we require, we have decided to proceed with the implementation of the project. A comparison between the different browsers as well as their support for key HTML5 API features our application requires is summarized in Table I. We have attached the version number as the feature set

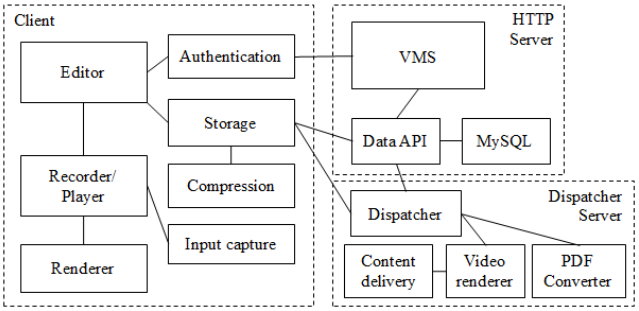

Fig. 2. Components in ShowNTell

available in a browser may change in later revisions of the browser.

B. Program Design

Our application uses a client-server model. Our client is designed to run within a web browser, and does not require the installation of any software aside from the browser itself, which is assumed to already be available on the user's device. The architecture used by ShowNTell is summarized in Figure 2.

The client is the front-end of our system, which is a website which the user can visit in order to use the services offered by our application. Using Canvas, we can implement whiteboard-like functions easily. We have also added other annotation tools such as highlighters and erasers which are tools available in similar software in other platforms. Browsers with Media Capture API support allow us to access the microphone on the user's device, allow us to capture audio for recording purposes. The Web Worker API which is commonly available on most modern browsers allows us to perform computationally expensive operations in the background, thus we usually perform data-related operation such as compression and conversion in workers.

The editor handles run-time drawing of visual data, as well as handles the state that will be displayed to the user. The editor includes the user interface such as the toolbar buttons, as well as the canvas drawing interface. The event recorder handles the capturing and replay of user input such as mouse coordinates. It also provides audio input capture if there is a microphone available. The storage component manages data belonging to the document and automated saving. The renderer renders the current document state to the user such as drawings, images, and textual objects.

On the server-side, we have two major servers, which are the HTTP and Dispatcher servers. The HTTP server hosts the web application for clients to access. The main scripting language used on this server is primarily PHP, which is a server-side language. The HTTP server also exposes external APIs for the uploading and retrieval of document data. The VMS (Video Management System) provides the front-end for users to manage their documents as well as share the documents with other users. Like typical HTTP servers, it uses

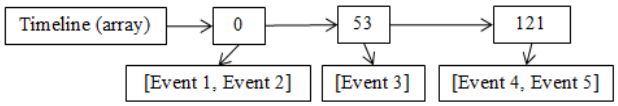

Fig. 3. Main timeline structure

normal LAMP stack which includes the Apache web server and MySQL database.

The dispatcher server is used to provide features that require long-running operations that should not be performed by the client due to its resource requirements. This server uses the JavaScript scripting language, and is run via NodeJS. The dispatcher server provides video rendering and PDF conversion capabilities.

IV. Implementation

A. Recording

During recording, audio from the microphone and user actions are captured simultaneously. In ShowNTell, user actions are referred to as events.

Events stored on the main timeline include slide change events and audio playback, which are typically global events that do not belong to any particular slides. Each document will only have one main timeline. During recording, events are added into the data structure representing the main timeline. Because recording involves time, time information has to be attached to each event. Playback requires an efficient means of traversing through events in the correct order.

Timeline data for the main timeline is represented in a 2-dimensional structure. The first dimension is the time, while the second is an ordered set of events occurring at that specific time. A visual representation of the timeline structure is shown in Figure 3.

During playback, we traverse across the time dimension, and process each event individually for each time slot. To reduce the amount of processing needed, each play-through will only go through the entire timeline once. Cumulative states are used to allow pause and resume without incurring significant costs. Fast forwarding of events is also efficiently supported.

B. Audio Recording

Audio capture is performed using the Media Capture API available in browsers such as Google Chrome and Mozilla Firefox. The API offers the ability to access raw audio PCM (Pulse-Code Modulation) data as it is generated [21].

In order to reduce the amount of time needed to save the document, we have chosen to upload audio as it is generated, rather than upload it at the end of the recording. Thus the audio is no longer a single contiguous file but instead distributed over multiple fragments. During recording, audio data is streamed to a buffer until a threshold is met. An interval of five seconds is chosen as the threshold as it offers a balance between the amount and size of fragments generated.

The number of fragments may increase complexity during playback and could also introduce additional overhead as each fragment is treated as an individual unit. However the size of a fragment should not be too large as it can incur additional computational requirements in processing of each fragment. When the threshold is met the buffer is encoded as a WAV file and uploaded to the server, after which the buffer cleared.

Not all browsers support the Media Capture API, thus we have developed a fallback which will involve the use of the Adobe Flash plugin. However, this fallback may not work on platforms that do not support the plugin, particularly mobile-based platforms. In the event whereby the application is unable to gain access to the microphone due to the software or hardware environment, the audio recording feature in ShowNTell will be disabled.

C. Audio Playback

Because we are pushing data as soon as it is available during recording, the audio recordings are not available as a contiguous audio stream but instead as fragments. In order to play the audio as a contiguous track, we have to perform audio chaining. Audio chaining involves preparing each audio to be played at some point in the future. The process of audio chaining is summarized below

1) Obtain list of fragments belonging to an audio collection
2) Playback is paused to buffer audio fragments in the collection
3) Relevant audio fragments are downloaded
4) Once sufficient fragments are downloaded, the fragments are arranged to play at a specific time
    a) Fragments whose end time precedes the current time marker are skipped.
    b) Fragments whose start time precede the current time marker, but end time does not will be played partially. These fragments will have their playback time set to the appropriate location, and played immediately.
    c) Fragments whose start time occurs after the time marker is to be played at a future time are queued at the appropriate time.
5) Once audio arrangement is done, playback is resumed

For smooth and seamless playback during audio chaining, each audio fragment should be encoded in a format that allows playback without any “gaps” or distortions. Formats such as MP3 and AAC may insert silence at the beginning of the audio file during the encoding process as compression artifacts [22]. Other formats such as Ogg and WebM do not have gaps as silence introduced as compression artifacts are already accounted for in the format. Lossless formats such as WAV will be gapless because there is no change in information in decoded audio data. We have used avconv which is part of libav [23] to perform audio compression on the server-side. The file sizes of the different formats are summarized in Figure 4.

The measured time (in milliseconds) to complete compression for a five second audio fragment is summarized in Figure 5. WAV is uncompressed data thus can be used as benchmark.

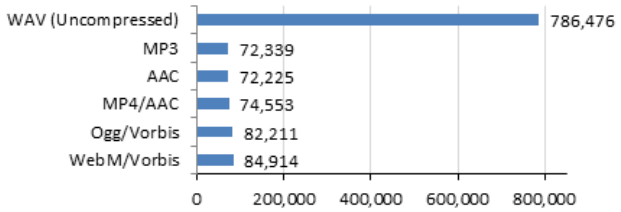

Fig. 4. Audio file sizes (in bytes)

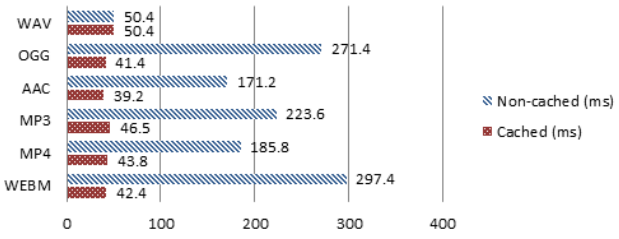

Fig. 5. Time taken (milliseconds) to compress and retrieve audio

Note that these timings do not include network transfer, thus while WAV takes the least processing time, the actual time taken to load a WAV file in the live environment could be slower than that of compressed files simply because compressed data is smaller in size. As compression of fragment can be done quickly, we can simply invoke the compression tool via PHP.

We have used caching to improve the server responsiveness during fetching of the compressed audio. Because audio data do not change, we can reuse data that was already compressed. Audio that have already been compressed are stored into the cache table in a similar manner as regular audio fragments, and retrieved when the relevant audio file is requested. Because the compressed audio fragments are small in size, storing them on the server will not require a large extra storage overhead.

D. Drawings

In order to generate drawings that appear smoother compared to the naive implementation (connecting points with straight lines), we have decided to employ a basic curve-fitting algorithm [24]. The following code snippet shows a portion of the curve drawing code that we have used when drawing a particular point:

```
if (thresholdmet) {
  var midX = (last.x + current.x) / 2;
  var midY = (last.y + current.y) / 2;
  context.quadraticCurveTo(last.x, last.y,
    midX, midY);
  last2.x = last.x; last2.y = last.y;
  last.x = current.x; last.y = current.y;
}
```

The above code generates a curve that passes near to the points, making it an approximation of a curve-fit. As we

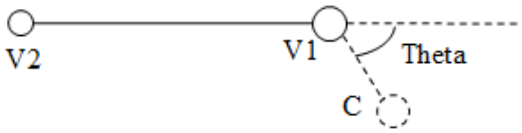

Fig. 6. Angle-based heuristic to account for changes in curve trajectory

are using Bezier curves[24], it will have the C-1 continuity property, giving a smooth appearance for the curve. Although the approximation introduces some small errors to the result, it still yields reasonable quality since the errors are usually small enough and not noticeable to the user. We are aware that there are algorithms that can produce curve-fits pass through all points however they will introduce greater complexity to the system [25]. Bezier curves require 3 control points. Hence, if there is only 2 points in the drawing we default to a straight line, and if there is only 1 point we simply render a dot.

Even with Bezier curves, the resulting curve still suffers from a "wavy" effect as a result of having points that deviate from the general path of the curve. These points are usually introduced due to vibrations during motion of the input device or by the user's hand. We have applied a simple de-noising algorithm to produce even smoother curves by eliminating points that do not contribute to the overall shape of the curve, but are significant enough that they introduce artifacts in the curve.

We have used two factors to determine whether a point should be rejected, these are Euclidean distance and the angle. If a point is very close to the last accepted point, it should be rejected as it is likely to not contribute much to the shape of the curve. An exception would be the case of sudden changes in curve trajectory, whereby we will need to keep the points as they contribute to the shape of the curve even though they are close to each other. We define a sudden change in trajectory as a significant change in the relative angles of the points in the trajectory (see Figure 6).

The last (V_1) and second last (V_2) points are points that have already been accepted in the curve. The candidate point (C) is the one which will be added. We measure the angle θ between the two vectors which can be obtained via dot-product between $\overrightarrow{V_2V_1}$ and $\overrightarrow{V_1C}$ unit vectors. A larger angle implies significant deviation in trajectory. We still apply a distance condition to prevent points from clustering near each other. The conditions for accepting a point are summarized below:

- First point (0 points): Starting point is always accepted
- Euclidean distance ($\geq$ 1 point): $|\overrightarrow{V_1C}| > 4$ pixels.
- Trajectory angle ($\geq$ 2 points): $cos^{-1}\frac{\overrightarrow{V_2V_1}\bullet\overrightarrow{V_1C}}{|\overrightarrow{V_2V_1}||\overrightarrow{V_1C}|} > 1$ radians, and $|\overrightarrow{V_1C}| > 2$ pixels.

Figure 7 illustrates the difference between the scenario where no de-noising is applied, and when de-noising is applied. The overall shape of the curve should be the same, although in the de-noised version there are less "wavy" patterns, particularly at gentler curves. The algorithm used

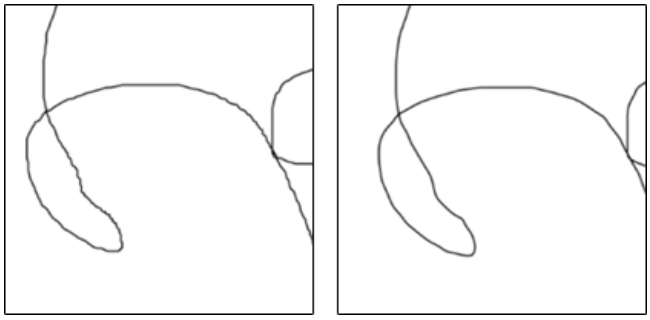

Fig. 7. Comparison between non-denoised (left) and noise reduction (right)

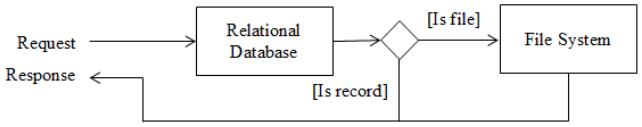

Fig. 8. Storage mechanism in ShowNTell

gave reasonably good quality drawings, and does not require expensive computation, allowing it to be deployed for the web browser.

E. Storage

Like most typical client-server and web-based solutions, data is stored on the server as the persistency of data on the client is not guaranteed. There are two common mechanisms for storing data for a client-server application, one of which is storing as records in a relational database such as MySQL, and the other is storing as regular files in the file system. As data is associated with a document, we represent this property as a relationship, thus the relational database was initially chosen as the storage mechanism. However, in practice we find that we can yield greater performance by storing larger records in the file system [26], thus we have adopted a hybrid storage mechanism which is shown in Figure 8.

Because we need to preserve relational information of all records in order for other components to perform lookups of document data, all units of data would have an entry in the database, including those that are stored on the file system instead of as a record on the database. If a unit of data exceeds a certain size threshold, it would be stored on the file system, thus we set a flag in the record to determine if the data returned would be from the database value or from the file system.

There will be some overhead when storing data corresponding to a record into a file, since the file must be created or overwritten on top of setting a flag in the database record. We observed that for file sizes less than 100KB, the difference in timings between the file system and database are minor and comparable to each other. The variance in timings is observed to be due to io-bound operations and network latency which affects both mechanisms. Beyond 100KB, we find that the database method will require significantly more time to

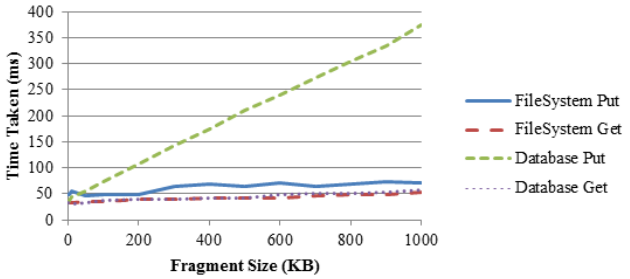

Fig. 9. Comparison between file and database storage

perform the put operation, and at sufficiently large sizes, the put operation would fail. We also observed that the get operations for both systems are comparable. The following experimental results were captured from our experiments. In the experiment we simulate a client-server interaction and measure the time taken (in milliseconds) to upload or retrieve fragments of different sizes (in kilobytes) from the browser to the server using both methods. The results are summarized in Figure 9.

For our system, we have set 128KB as a threshold for determining which storage system to use. Small records such as metadata are stored in the database. Records larger than 128KB, which are typically audio fragments, images, and timeline data, are stored into the file system.

F. Other services

There are some services that do not run within the web browser, but instead performed on the server as it requires significant computational resources. These are usually handled by the dispatcher server, which provides video rendering services. Our video renderer is written using JavaScript and runs as a NodeJS application. When a user requests for a document to be rendered, the document ID is passed to the dispatcher service, which spawns the video rendering process. The video rendering process performs the following actions:

1) Download the document chunks.
2) Render the timeline data into a video frame-by-frame, using avconv (from libav) “pipe2image” feature. Each frame is rendered as a JPEG using node-canvas. The intermediate video (without audio) generated uses the MKV format.
3) Combine the audio chunks into a contiguous audio file. We are using SoX to perform the splicing of audio. The intermediate audio file generated uses the WAV format. The audio file may be re-encoded to 44100 sample rate to ensure that all fragments have the same sample rate.
4) Combine the generated video and audio into a single video file. The resultant file uses the MP4 (H.264) format. We are using avconv to combine the files.
5) Upload the final video file to the CDN (Content Delivery Network).
6) Send an email to the owner of the document informing them that the video has been rendered completely.

When rendering the frame, we have chosen JPEG as it provides a smaller sized image while retaining image quality, reducing the amount of time needed to transfer the image from our renderer to avconv. We have used WAV as the audio input since WAV files are generated by our client application. The intermediate video format is MKV as we find that the generated video track is of good quality, and does not suffer from significant compression artifacts when re-encoded to MP4 in the final stage. The resultant video format is MP4 with H.264/AAC encoding as it is supported by most od the modern CDNs and our CDN of school's Integrated Virtual Learning (IVLE) platform [27].

In addition the the video renderer, ShowNTell supports PDF documents which can be imported as slides used by ShowNTell. Document conversion uses the imagemagick tool which has support for PDF to image conversion [28]. The output image chosen is PNG as it offers good image quality.

V. Evaluation

In this section, we highlight the usability, performance and scalability of ShowNTell system. The evaluations include analysis of user feedback on our system, as well as other evaluations we have drawn from our testing of the system.

A. User Study

We conducted field tests on our application with users. Most of these users came from the Faculty of Computing, although there were others from the Faculty of Science of our University. In this initial study, students and instructors from the following modules tried the system: PC1431 (Physics IE), CG3204L (Computer Networks) and CS3247 (Game Development). Our user study primarily focuses on participants from the CS3247 module.

During the March of 2015, we allowed students from the class of CS3247 to complete their assignments using our implementation of ShowNTell, thus allowing us to gather some information on user behavior, preferences, as well as the readiness of our system in terms of scalability. Altogether there were over 40 participants. The experiment was conducted over a period of one week, thus the load on the system could be confined and evaluated during this period. Prior to the student study, we had also opened up the system to other educators who were interested in our implementation so as to make any adjustments to the system if needed.

During this experiment, participants were to complete a recording to answer two technical questions of the module assignment using ShowNTell tool. The use of slides was required for the assignment, and was limited to 10 slides, thus it was expected that most students would attempt to use the PDF import feature. As the presentation was to be delivered orally, the students must also use the audio recording feature. There was an original requirement to use the video export, but that was lifted due to technical issues experienced towards the end of the experiment. The expected recording duration was at most five minutes. Upon completion of the experiment, the students were asked to fill in a feedback form.

TABLE II
User Study Demographics

Participants	45 (30 respondents)
Participant type	Students (97%) Academic Staff (3%)
Browser used	Google Chrome (80%) Mozilla Firefox (17%) Safari (7%)
Recording duration	<3 minutes (7%) 3 to 5 minutes (49%) 5 to 10 minutes (42%) >10 minutes (2%)
Input used	Mouse (93%) Touch (40%) Stylus (50%)

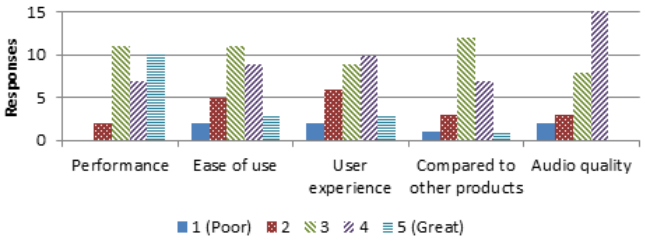

Fig. 10. User responses to various aspects

B. Evaluation Results

The survey primarily covered on user demographics as well as their preferred methods of using the system. Other feedback questions were also designed so as to inspire future iterations of our project. The survey results are summarized in table II. Majority of the users used ShowNTell on Google Chrome for Windows. Most of the participants were familiar with other video recording solutions, in particular Camtasia, Camstudio and OBS. The most common recording duration created by users was approximately five minutes long. The most commonly used input device was the mouse, as many students were using ShowNTell on a computer instead of a mobile device.

In general, most participants (76%) find our implementation easy to use and adequate (see Figure 10). The participants have indicated that ShowNTell provided sufficient capabilities to complete the requested assignment, although most have indicated the system could be improved further. Participants have indicated that ShowNTell is usable with the mouse and stylus, although participants who have used touch-based inputs mentioned them to be less comfortable. We hypothesize that touch-based input is harder to draw objects accurately for extended periods of time compared to a stylus or mouse.

Most participants (93%) have indicated that the performance of ShowNTell is adequate. Playback, recording, and saving of documents were perceived to be completed within reasonable time, fulfilling the main objective of our project which is to implement a responsive and usable recording solution in a web browser. Most actions on a recording can be completed within seconds, without the user needing to wait too long before

getting a response.

In terms of features, many participants (43%) find the PDF import function to be the most useful feature. It is observed that most participants (97%) have used the feature to import their slides, allowing them to quickly create the materials for a recording. Users have also found the sharing functions useful and easy-to-use.

It is observed that a number of documents (32%) have no additional drawings/image annotations. As the use of slides coupled with audio overlay is sufficient to complete the assignment, we suppose that these users do not use the annotation functions simply because it is not needed to complete the task.

Among those who have tried adding annotations, there were some who have found it frustrating to use the feature, particularly those using only a mouse-based input, thus only use the annotations in a limited way. For users who have tried a stylus or touch-based input, their rating is split between easy-to-use and challenging-to-use. We find that most users who have prior experience with recording solutions that do not have annotation capabilities (such as Camstudio) are more likely to indicate a negative experience with annotations.

Many users have suggested non-linear editing of the recordings. While we have evaluated the idea of introducing non-linear editing to ShowNTell, we find that it might be too technical challenging to implement it in the remaining time frame, thus we would leave this challenge as a form of future work that others could take up.

C. Evaluation Conclusions

We find that ShowNTell is a reasonably good implementation of a browser-based recording solution. However, we find that most users prefer to use it as a simple recording tool that overlays audio on top of slides. Nonetheless, we find that ShowNTell offers sufficient capabilities to act as an eLearning tool, and thus is capable of satisfying the tasks outlined in the experiment. There are many possible improvements that could be done to improve the usefulness of ShowNTell, such as support for rewriting past recordings.

Most users indicated that ShowNTell is responsive, implying that our performance optimization techniques and management of data is adequate for our solution to deliver a responsive experience within a web browser. Users were able to create recordings up to 10 minutes or longer, showing that ShowNTell is capable of supporting recordings of sizeable duration.

An Individual educator's evaluation statement: "The application is very good for the modules such as physics and mathematics which have lots of equations and drawings. I have used the import pic, free hand drawing and writing using stylus, highlighting, voice over recording and video rendering to answer students query. All these features makes the explanations very clear, with minimum text usage, saves time to input equations and symbols and brings it to a level which is comparable to face to face consultations with the students".

VI. LIMITATION

As some browsers have not implemented the APIs used by ShowNTell, the application is not expected to work on all browsers available in the market. However, as the application uses standards compliant APIs that are in the W3C draft specifications, the application will eventually be supported by browsers in the future that eventually implement those specifications.

At present, ShowNTell works well on Google Chrome and Mozilla Firefox. For browsers that only support a subset of the APIs required by ShowNTell, we may opt to use a fallback mechanism or simply disable the feature that uses the API. Due to performance limitations particularly in the mobile environment, certain techniques such as client-side compression might not be feasible in practice thus we use alternative approaches like server-side compression to obtain a similar result. However, this limitation is expected to ease with better hardware in the future.

ShowNTell is currently supported on most browsers for desktop and notebooks, which includes Internet Explorer, Google Chrome, Mozilla Firefox, and Safari. For Internet Explorer and Safari, we have taken steps to address the lack of support for certain APIs by providing a fallback, such as in the case of audio playback and recording.

On Android, ShowNTell is supported by Google Chrome and Mozilla Firefox. Among these browsers, we find Google Chrome to be more stable in performance compared to Firefox. Other browsers on Android such as the stock browser is not capable of supporting ShowNTell as it lacks many of the required APIs needed by many basic functions in ShowNTell.

On iOS, ShowNTell is partially supported by Safari and Chrome. As there are no browser-based APIs for microphone access in these browsers on iOS, ShowNTell will not be able to record any audio on those platforms. Nonetheless, other features in ShowNTell are still supported, thus users can still watch documents on these devices.

VII. CONCLUSION

We have found that it is feasible to implement a recording solution for the web platform that is supported on most web browsers. We have also presented solutions that can enable a fairly responsive application even when manipulating sizeable amounts of data. User studies have indicated that ShowNTell is comparable to other similar tools in the industry.

Overall ShowNTell works well in providing the features of a standard whiteboard recording system over the browser, and provides sufficient features to accomplish the basic requirements for its purposes in E-Learning. As ShowNTell is supported across multiple platforms and does not require installation of any software, it poses a low barrier to entry for students and instructors to use it, making e-learning more effective. ShowNTell is continuously evolving. Future releases will include complete video editing and real-time collaborative content editing modules to further enrich the learning experience.

REFERENCES

[1] ShowNTell, "Shown tell," http://www.sntboard.com/, National University of Singapore, 2015, (Accessed on Apr 25, 2015).

[2] W3C, "Html canvas 2d context," http://www.w3.org/TR/2dcontext/, World Wide Web Consortium, 2014, (Accessed on October 21, 2014).

[3] D. Manh Hung, "Coursemology: A gamified online education platform," National University of Singapore, 2012.

[4] T. Xiao, "Coursemology: A gamified online education platform," National University of Singapore, 2013.

[5] K. Haramundanis, "Why use screen captures? an experience report." New York, NY, USA: ACM, 2011.

[6] CamStudio, "Camstudio, free screen recording software." http://camstudio.org/, CamStudio, 2014, (Accessed on October 21, 2014).

[7] TechSmith, "Camtasia, screen recorder and video editor." http://www.techsmith.com/camtasia.html, TechSmith, 2015, (Accessed on March 10, 2015).

[8] OBS, "Open broadcaster software," https://obsproject.com/, Open Broadcaster Software, 2015, (Accessed on March 10, 2015).

[9] Microsoft, "Bitblt function (windows)," https://msdn.microsoft.com/en-us/library/dd183370%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396, Microsoft, 2015, (Accessed on January 21, 2015).

[10] EyePowerGames, "Ink2go," http://ink2go.org/, EyePower Games, 2014, (Accessed on October 21, 2014).

[11] A. Noe, "Avi file format," http://www.alexander-noe.com/video/documentation/avi.pdf, 2006, (Accessed on March 10, 2014).

[12] Beepa, "Fraps - real-time video capture and benchmarking," http://www.fraps.com/, Beepa Pty Ltd, 2015, (Accessed on March 15, 2015).

[13] Android, "Manifest.permission," http://developer.android.com/reference/android/Manifest.permission.html#READ_FRAME_BUFFER, Google, 2015, (Accessed on March 31, 2015).

[14] ShowMe, "About the showme online learning community," http://www.showme.com/about_showme/, ShowMe, 2014, (Accessed on October 21, 2014).

[15] MorrisCooke, "Explaineverything," http://www.morriscooke.com/?p=134, MorrisCooke, 2014, (Accessed on October 21, 2014).

[16] BigBlueButton, "Overview, bigbluebutton," http://bigbluebutton.org/, BigBlueButton, 2015, (Accessed on October 21, 2014).

[17] S. Jobs, "Thoughts on flash," http://www.apple.com/hotnews/thoughts-on-flash/, Apple, April 2010, (Accessed on March 31, 2015).

[18] S. Sinosky, "Metro style browsing and plug-in free html5," http://blogs.msdn.com/b/b8/archive/2011/09/14/metro-style-browsing-and-plug-in-free-html5.aspx, Microsoft, September 2011, (Accessed on March 31, 2015).

[19] Adobe, "Flash to focus on pc browsing and mobile apps; adobe to more aggressively contribute to html5," http://blogs.adobe.com/conversations/2011/11/flash-focus.html, Adobe, November 2011, (Accessed on March 31, 2015).

[20] A. Barstow, J. Kostiainen, Anssi; Rabin, J. Manrique Lopez, M. Lamouri, M. Caceres, F. Daoust, and R. Cremin, "Standards for web applications on mobile: current state and roadmap," pp. 5,11–13,28,48, 2014, (Accessed on March 10, 2015).

[21] W3C, "Media capture and streams," https://w3c.github.io/mediacapture-main/getusermedia.html, World Wide Web Consortium, 2014, (Accessed on March 29, 2015).

[22] M. Taylor, "Lame technical faq," http://lame.sourceforge.net/tech-FAQ.txt, June 2000, (Accessed on March 10, 2015).

[23] libav, "Open source audio and video processing tools," https://libav.org/, libav, 2015, (Accessed on March 15, 2015).

[24] K. Peter, *Foundation ActionScript 3.0 animation: making things move*. New York: Apress, 2007.

[25] H. Akima, "A new method of interpolation and smooth curve fitting based on local procedures." New York, NY, USA: ACM, October 1970.

[26] R. Sears, C. Van Ingen, and J. Gray, "To blob or not to blob: Large object storage in a database or a filesystem." Microsoft, 2006.

[27] "NUS", "Integrated virtual learning environment," https://ivle.nus.edu.sg/, National University of Singapore, 2015, (Accessed on Apr 26, 2015).

[28] ImageMagick, "Imagemagick: Convert, edit, or compose bitmap images," http://www.imagemagick.org/, ImageMagick, 2015, (Accessed on March 15, 2015).