The MIT Unified Submission and Grading System (MUGS)

Shannon Hwang, Jason Paulos

1. Introduction

6.033 is an MIT class that covers the fundamentals of computer system engineering and design, yet its grading system is severely lacking. Currently, several key system functionalities are haphazardly split between a submission site, Gradescope, and staff personal machines. 6.033's current system also fails keep track of files before submission or facilitate group collaboration and submission for the team assignments that make up nearly half of the course.

We present an updated system design, the MIT Unified Submission and Grading System (MUGS), that solves these problems while prioritizing four hierarchical design goals: reliability, security, simplicity of use, and flexibility. We prioritize system reliability above all other goals because we believe that a file and grading system is essentially useless to its users if it cannot reliably store files and grades; we achieve reliability by keeping unique records of all files and submissions and ensuring reliable file transmission across unreliable networks. Next, we prioritize security since allowing only appropriate students to view grades and assignments is vital to the ethical and normal operation of any class using MUGS. We achieve security by defining a hierarchy of user roles and permissions, which ensures that files are only accessible by the correct users, and by encrypting transmitted data. We achieve simplicity of use – which we believe is necessary for students and staff to adopt and use the system with minimal complaint – by having users interact with the system only through a user interface. In addition, we build users, groups, and assignment directories into the file system structure, create convenient links between directories for users to access their relevant directories, automate time-consuming processes, and base system security and functionalities on the file system structure. Finally, we achieve flexibility, which is important for potentially adapting the system to other class structures, by utilizing an abstract user, group, and assignment structure that can be easily adjusted to the course structure of other classes.

2. System Overview

As shown in Figure 1, our system consists of individual components that communicate with each other to execute varied user requests. Some components reside on the MUGS Server, while others are contained in their own servers and accessed over the network.

The MUGS back-end, located on the MUGS Server, is our system's central hub. It receives users' requests from the front-end, determines which actions to take, and executes these actions on the MIT File System. All of our system's custom logic resides here.

The MIT File System (MFS), also running on the MUGS Server, stores all persistent information in our system. This includes user information, assignment submissions, grades, and more. MFS' permission system contributes to system security by determining which actions users can take.

The MUGS front-end is the primary interface where users interact with our system. The front-end is responsible for passing user authentication information securely to the MIT ID service (MIDS), which authenticates users by granting them a Kerberos ID. This Kerberos ID is included in requests to the MUGS back-end to identify users and their requests. The front-end is also responsible for uploading and downloading files to an instance of MFS using the MIT Sync Service (MSS).

Additionally, the MUGS back-end and MSS use the MIT Lock Service (MLS) to coordinate concurrent access to MFS. This is described in further detail in Section 2.5.

Gradescope interacts with both end users and the MUGS back-end. Students and graders submit and grade assignments on Gradescope, respectively, and the MUGS back-end synchronizes Gradescope grades with MFS.

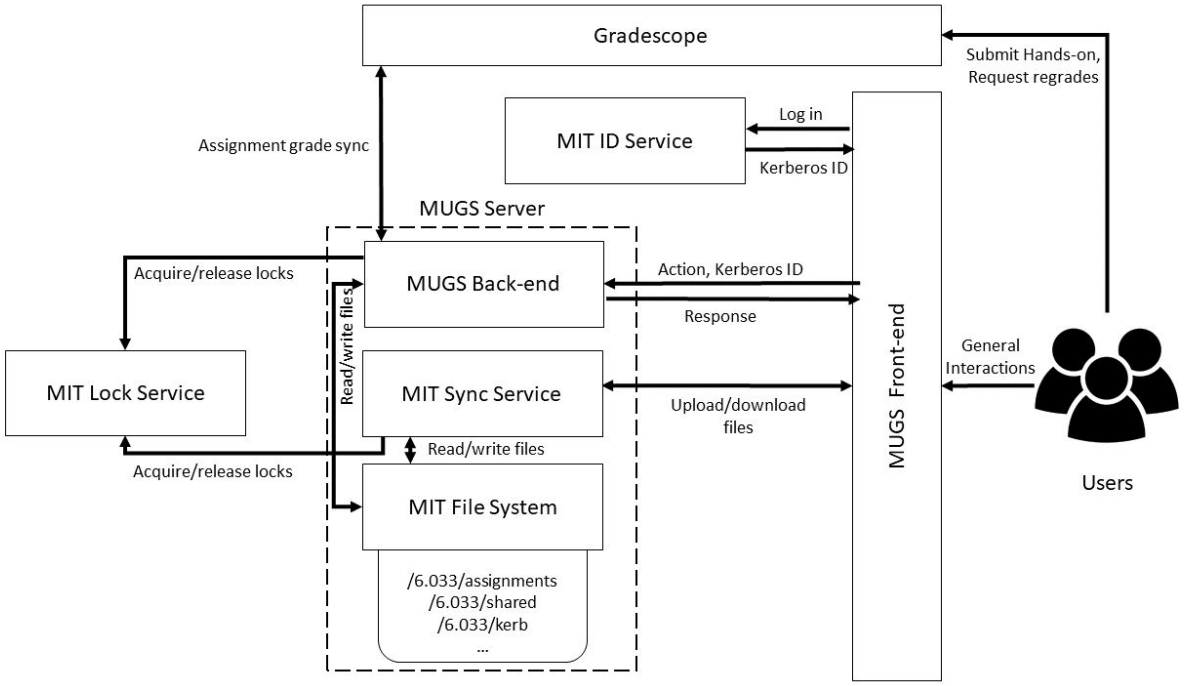

Figure 1: An overview of the different components and interactions in MUGS. Users interface with a front-end, which handles routing requests to the MIT ID Service, MIT Sync Service, and MUGS back-end. The MUGS back-end, MIT Sync Service, and MIT File System run on the MUGS Server. The MUGS back-end and MIT Sync Service also share access to the MIT Lock Service, which coordinates their use of the MIT File System. Gradescope has interactions with both the users and MUGS back-end.

2.1 Roles

We use Kerberos IDs to represent every student and staff member. In order to simply implement security policies, we also assign a group ID to various groupings that arise naturally in the course such as recitations, tutorials, and teams, organized using a hierarchical model that reflects the various levels of permissions that the groups naturally have. With group and user

IDs, controlling security and user access to various files and directories simply involves setting permissions in the file system corresponding to different user IDs. Figure 2 shows an example hierarchy of users and groups in the course. Notice that members of a group can be specific users as well members of other groups.

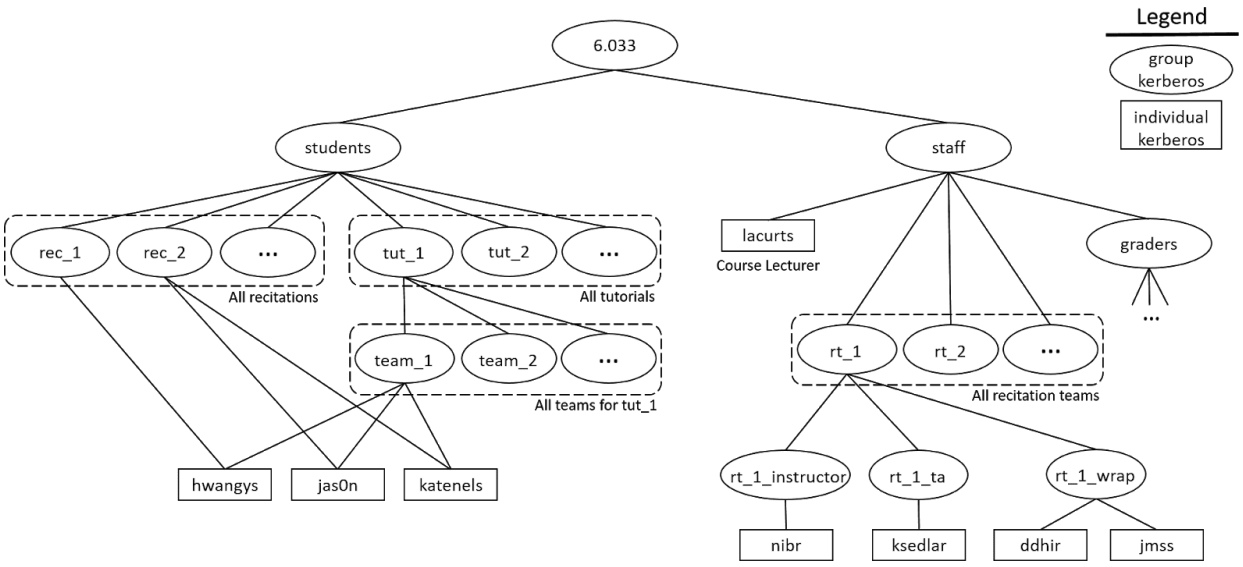

Figure 2: The hierarchy of users and groups in MUGS, shown with specific users as an example. 6.033 is the common ancestor of all roles, splitting users into the students and staff groups.

2.2 Users

A *role* in our system can be either an individual user or a group. Every role has a unique Kerberos, which is essential for identification in our system. The *parents* of a role are all of the groups to which it immediately belongs. Note that 6.033 is the only role that does not have a parent, and individual students have more than one parent. The *family* of a role are all of the groups to which it belongs at any depth, and itself. For example, in Figure 2, the parents of `jas0n` are `rec_2` and `team_1`, and the family of `jas0n` is `jas0n`, `rec_2`, `team_1`, `tut_1`, `students`, and `6.033`. A user's family is an essential part of the permissions of our system.

2.3 Permissions

An individual can use their Kerberos to perform actions on behalf of any members of their family. This functionality is implemented by examining all Kerberos belonging to the user's family. The system will attempt to use each Kerberos to complete the action until either it succeeds in finding a Kerberos that can perform the action, or every family member's Kerberos is tried (in which case the action fails). In order to find the family of a user, the system searches the group tree using breadth first search upward from the user's Kerberos. This is done by following parent links for each role inside of its home directory, described in further detail in

Section 4. The parent links are evaluated in lexicographical order at each level, thus guaranteeing that the system will always iterate over a user's family in the same order.

We chose to prioritize flexibility when designing the permission system because we believe the ability to represent an arbitrarily complex grouping system with shared permissions is an essential part of any course structure. This flexibility comes at the expense of simplicity of implementation, since trying every role in a user's family is more complicated than just trying one. However, it simplifies security for the rest of the system, which does not have to reimplement group lookup in order to use permissions. Implementing permission verification in only one part of the system increases security by reducing the likelihood of bugs that could occur if each subsystem implemented its own version of permission verification.

2.4. File System

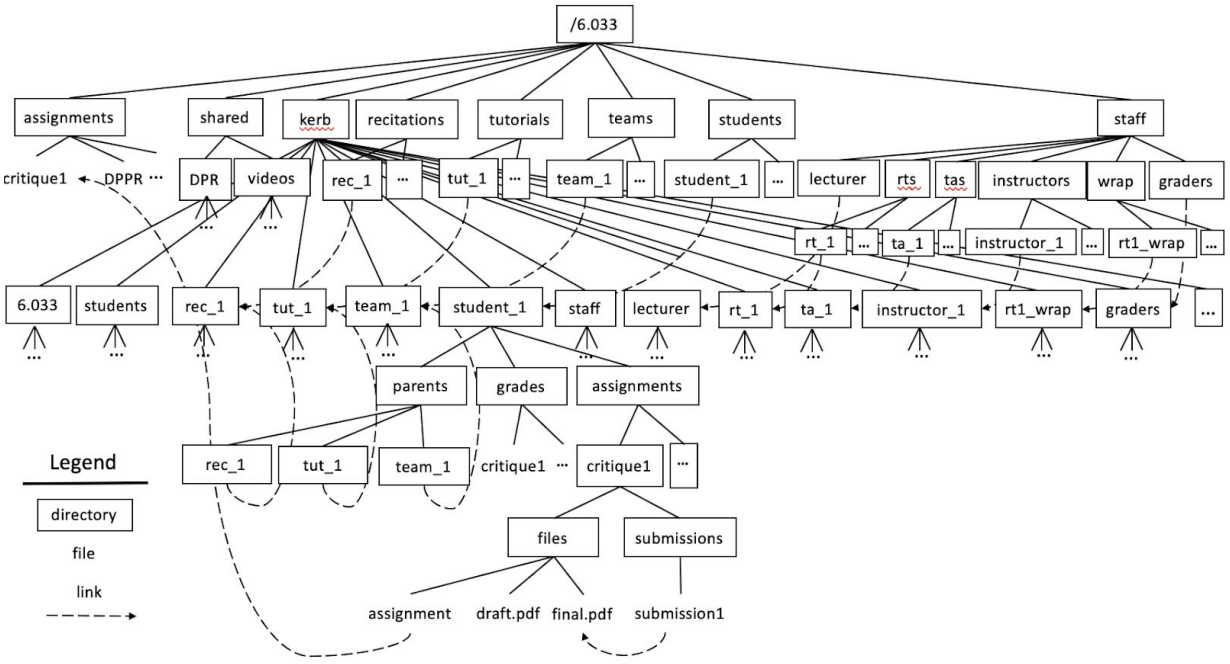

Figure 3a: A subsection of the MUGS file system. The directory `kerb` contains the home directories of all users. Symbolic links are used throughout the file system to link synonymous entities and ensure information is accessible to multiple users in well-defined locations. The structure of unexpanded directories is specified in **Figure 3b**.

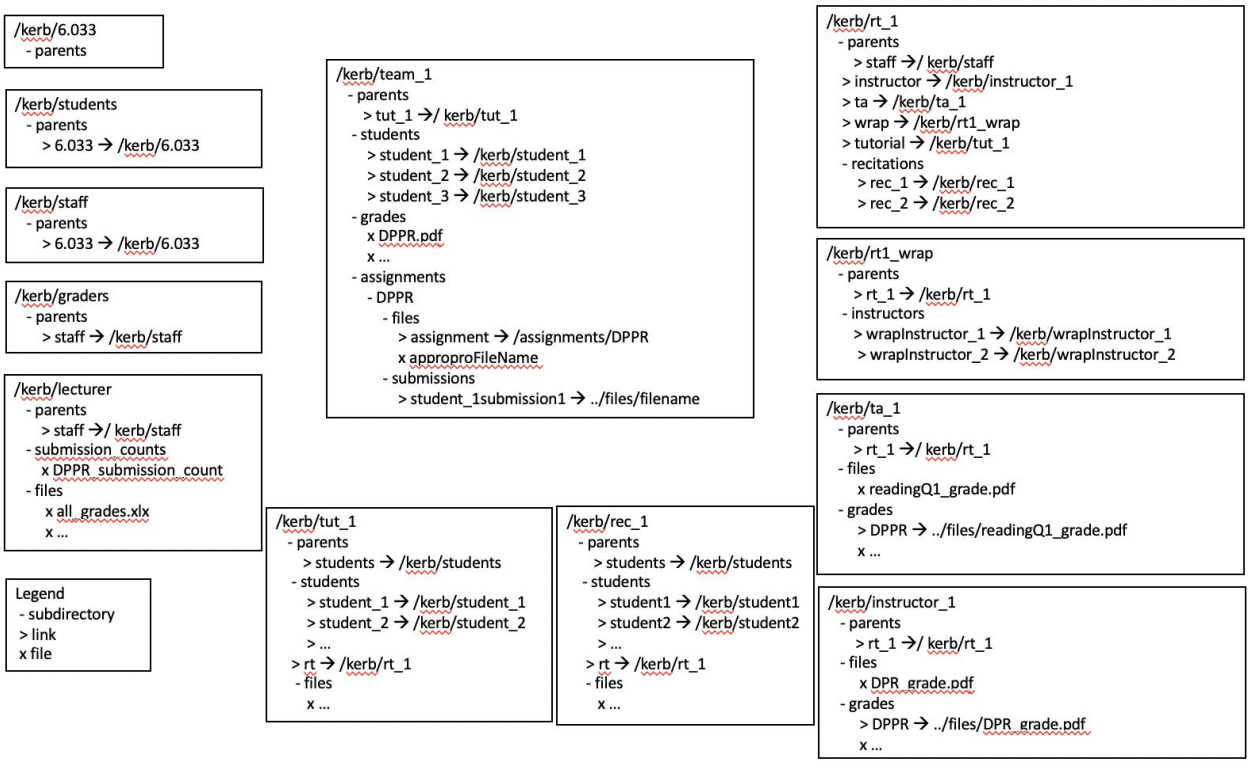

Figure 3b: Directory layout of sample directories in the MUGS file system.

All of MUGS' persistent information, such as information about teams, assignments, and grades, is stored on or encoded in the structure of the MIT File System.

2.4.1 High Level Structure

The file system root folder, `/6.033`, contains all of the data required for MUGS to operate. This folder contains the folders `assignments`, `shared`, `kerb`, `recitations`, `tutorials`, `teams`, `students`, and `staff`, each of which is described below:

`/6.033/assignments` contains assignment description files (further described in Section 2.4.3) for each of the assignments in the course.

`/6.033/shared` is a general-purpose location for the course lecturer to share files with the entire class. It and its contents are writable only by the course lecturer and readable by everyone. It contains folders DPR and `videos` that store links to each team's submitted design project proposal for peer-review and their video for students to vote on. It would also be a good place for the course lecturer to upload lecture notes or other course resources.

`/6.033/kerb` contains the home directory for each MUGS user, which encompasses individual students and staff as well as groups of users as described in Section 2.2; home directories are located at `/6.033/kerb/userKerberos`. Home directory contents vary between users, as

described in Section 2.4.2, but every user home directory must contain a `parents` folder which indicates to which groups the user belongs. A user *x* belongs to group *y* if and only if `/6.033/kerb/`*x*`/parents/`*y* exists and is a symbolic link to user *y*'s home directory. Because the contents of each `parents` folder completely determines the set of permissions a user can use in MUGS, the `parents` directory is only readable and writable by the course lecturer.

The following directories link to users' home directories in different ways in order to provide easy access to specific groups:

`/6.033/recitations` contains symbolic links to all recitation sections' home directories and the directory `unassigned`, which contains links to the home directories of students who have not been assigned to a recitation.
`/6.033/tutorials` contains symbolic links to all tutorial sections' home directories and the directory `unassigned`, which contains links to the home directories of students who have not been assigned to a tutorial.
`/6.033/teams` contains symbolic links to all teams' home directories and the directory `unassigned`, which contains links to the home directories of students who have not been assigned to a project team.
`/6.033/students` contains symbolic links to all students' home directories and the directory `dropped`, which contains links to the home directories of students who have dropped the course.
`/6.033/staff/lecturer` is a symbolic link to the course lecturer's home directory.
`/6.033/staff/rts` contains symbolic links to all recitation teams' home directories.
`/6.033/staff/tas` contains symbolic links to all TAs' home directories.
`/6.033/staff/instructors` contains symbolic links to all recitation instructors' home directories.
`/6.033/staff/graders` contains symbolic links to all graders' home directories.

2.4.2 Home Directories

The home directory of each role depends on its specific needs. Here we describe the cases where home directories contain more than just parent links.

2.4.2.1 Students

Student home directories contain additional directories in their home directory: `team`, `rec`, `tut`, `grades`, and `assignments`. `team`, `rec`, and `tut` are all symbolic links to the student's team, recitation, and tutorial home directories, respectively. `grades` contains a subdirectory for each assignment, which in turn contains grade files for that assignment. Each grade file is given the name of the staff role responsible for grading that assignment, and these files are only writable by that role. Before grades are released for an assignment, the `grades` directory for an assignment is empty, and after grades are released the system creates symbolic links in this

directory to the grade files in the appropriate staff directories. `assignments` also contains a subdirectory for each assignment, which in turn contain two more subdirectories, `files` and `submissions`. `files` contains intermediate files that the student can upload to store work in progress for that assignment. `submissions` contains symbolic links to some files in the `files` folder, indicating which ones have been submitted for the assignment. The name of the links in the `submissions` folder contain the timestamp of the submission, which is used to determine if the submission was turned in on time and the order of submissions.

2.4.2.2 Teams

Team home directories contain the additional directories `students`, `grades`, and `assignments`. `grades` and `assignments` function the same as in student home directories, except they contain assignments that must be submitted as a team. `students` contains symbolic links to each team member's home directory. It can be used to quickly look up which students are members of the team.

2.4.2.3 Tutorials and Recitations

Tutorial and recitation home directories contain the additional directories `students` and `rt`. `Students` contains symbolic links to each of the tutorial section's students. `rt` is a symbolic link to the home directory of the recitation team group that teaches the tutorial or recitation.

2.4.2.4 Recitation Teams

Recitation team home directories contain the additional directories `instructor`, `ta`, `wrap`, `tutorial`, and `recitation`. `instructor`, `ta`, and `wrap` are symbolic links to the recitation team's recitation instructor, TA, and WRAP instructors' home directories, while `tutorial` and `recitation` are directories that contain symbolic links to the tutorial groups and recitation groups in the recitation team.

2.4.2.5 Recitation Instructors and TAs

Each of these users' home directories contains the additional directory `rt`, which is a symbolic link to the recitation team to which they belong.

2.4.2.6 WRAP Instructors

The WRAP instructors' home directories contains an `rt` directory linking to their recitation team as well. Additionally, since multiple instructors may be WRAP instructors for a given tutorial, there is an `instructors` folder which contains symbolic links to the home directories for each of the WRAP instructors for the section.

2.4.3 Assignments

The `/6.033/assignments` directory contains an assignment description file for each assignment to let the system track the different assignments in 6.033. Each assignment descriptor file is formatted as a simple key-value mapping of important assignment properties. These properties are as follows:

- **Name:** the name of the assignment
- **Hand in procedure:** whether the assignment should be submitted through MUGS, Gradescope, or not at all, in the case of an exam.
- **Type:** whether the assignment is for individual students or design teams.
- **Due date:** a timestamp describing the date and time the assignment is due.
- **Graders:** who is responsible for grading and/or giving feedback about the assignment. This includes relations such as a student's TA, recitation instructor, WRAP instructor, or Gradescope. Multiple people can included here, with specified weights to their individual grades. A weight of 0 indicates that a grade is not required from the individual, only comments.
- **Points:** the number of points this assignment is worth.
- **Late penalty script:** a Bash script that takes as an input the time of submission and prints the number of points to subtract as a late penalty. This script is run when the assignment is graded to calculate the late penalty that submission should receive. By convention, the script should not take off any points if the assignment is submitted before the due date, although MUGS is agnostic of this.
- **Validation script (optional):** an optional Bash script that takes as an input the filename of a submitted assignment. The script can then read the file and determine whether it is a valid submission or not. If the submission is valid, it should terminate with exit code 0. Any other exit code indicates that the submission is not valid and cannot be accepted by MUGS. If this value is not indicated for the assignment, MUGS will assume every submission is valid.

If included, the validation script runs every time a student submits a file for this assignment. Their submission is pending until the script returns. If the script indicates the submission is invalid, the submission is not accepted by MUGS and the student is notified of this. Otherwise the submission is accepted successfully.

These assignment files can only be modified or created by the course instructor, who can do so by through a page in the MUGS front-end dedicated to creating and modifying assignments. In order to create an assignment, the system creates a subdirectory in every student/team's `assignment` folder as described in Section 3.3.3. After an assignment is created, the instructor can modify all properties except for the name, hand in procedure, type, and graders, since these properties modify the file system immediately after creation and are difficult to undo. We believe this is an acceptable limitation, since this solution allows for the points, due date, late penalty, and validation script to be modified, which are much more prone to change.

The use of instructor-uploaded Bash scripts for calculating late penalties and submission validation makes assignment descriptions very flexible and powerful. They allow arbitrarily complex operations to be run on submissions, and since they are completely specified by the course instructor, it is relatively easy to test and debug these scripts. For instance, if the late policy was hardcoded into MUGS, it would be much more difficult to find errors and make changes. This bash script can only support assignments whose late penalties rely only on the time of submission, such as a linear loss of points or a step function, and is not designed to support classes where students request late days before they submit their assignments, or where late days are algorithmically distributed among assignments to minimize grade impact. Thus, our system's support of late policies sacrifices system flexibility for ease of use and reliability. We believe this is acceptable because system flexibility is our lowest-priority design goal.

2.5 Locks

MUGS uses the MIT Lock Service (MLS) to manage concurrent access to the file system via read and write locks. All locks for files are acquired on the file's absolute path, with all symbolic links resolved to their actual values. Whenever the system needs to read a file, it acquires the read lock on the file, and whenever the system needs to write to a file, it acquires the write lock on that file. Additionally, if multiple locks need to be held simultaneously, they are acquired and released by the system in lexicographical order of the file's absolute path. This pattern prevents deadlocks from happening when implemented correctly.

We chose this locking strategy in order to support simplicity of use and reliability for our system. We believe having a read/write lock system for file access is easy to understand and correspondingly decreases the probability of running into locking problems. This comes at a performance tradeoff, as each individual file is protected by a lock that limits the rate of access to each file. We believe that the guarantee of always successfully writing or reading a complete file outweighs the performance penalty of sometimes waiting for a lock to be released.

2.6 Syncing

We utilize the MIT Sync Service (MSS) to upload and download files to MUGS. MSS allows individual users to upload files from their machine to any directory they can create files in, as well as download any accessible files to their machine. We use MLS to handle concurrent accesses to files in MFS as described in Section 2.5.

2.6.1 Uploading

When a user requests to upload a file to MUGS through the front-end, the system first checks if that user has permission to write to the indicated file. If not, the transfer fails immediately. In the case that they do have sufficient permission, the client begins sending the file. MSS writes this stream of data to a new temporary file in the system's `/tmp` directory. If the transfer is stopped

or fails before finishing, the temporary file can be deleted with no consequences. When all of the data is received and stored in the temporary file, MSS renames the temporary file to the indicated destination file for the transfer, overwriting partial writes to the file if it already exists. Using a temporary is advantageous for this process because it allows the system to recover from a failed transfer without having to undo partial writes to files; because each file is uniquely named, there is no danger of rewriting previous uploads from other users. We believe this contributes to the reliability of our system, since it can tolerate failed uploads easily.

2.6.2 Downloading

Downloading a file is more straightforward than the uploading process. When a user requests to download a file, the system checks to see if the files exists and if the user has sufficient permission to read the file. If either of the previous conditions are false, the download fails. Otherwise, the MSS streams the file to the client. Since downloading is a read-only action, if the download fails or the user stops early, the client can simply request the same download again with no harmful consequences.

2.7 Network Protocol

Students in 6.033 use MSS to transfer files to our system. These files often contain sensitive information, such as a student's personal work or grades. It would be disastrous if anyone other than the student saw these transfers, or was able to modify them. Using HyperText Transfer Protocol Secure (HTTPS) as our network protocol provides a reliable and secure way to transfer files.

2.7.1 Usage

To upload a file through MSS, a client initiates an HTTPS POST request to the MUGS server on the socket reserved for MSS. The POST request's location contains the remote filename and the request's data contains the uploaded file's contents. MSS responds with HTTP status code 201 (Created) once it successfully receives all data.

To download a file through MSS, a client initiates an HTTPS GET request to the same server and socket described above. The GET request's location contains the remote filename of the file to download. If there are no issues, MSS responds with the HTTP status code 200 (OK) and streams the file's contents to the client.

In both cases, the client's Kerberos ID is included as a request parameter so that MSS can authenticate the user. If authentication cannot be performed, MSS responds with HTTP status code 401 (Unauthorized), and if the user does not have permission to read or write to the indicated file, MSS responds with status code 403 (Forbidden).

HTTPS notifications about the status of a student's file transfer increases our system's reliability, as student's files will not mysteriously go missing during transfer.

2.7.2 Security

HTTPS is an extension of HTTP with added security features like server authentication and request/response encryption. Server authentication allows clients to verify that the machine they connect to is actually the MUGS server so that a man in the middle attack is not possible. Additionally, request/response encryption ensures that all data transmitted over the network is unreadable by any third party observers.

In order to support HTTPS, MSS needs to be issued an SSL server certificate from a Certificate Authority (CA) trusted by the client. We recommend that the system administrator obtains this certificate from DigiCert before the semester begins and that the certificate is not set to expire until after the semester ends. This will avoid the challenge of updating certificates during the semester and potentially reducing the availability of MUGS. We chose DigiCert as a CA because web.mit.edu uses it to support HTTPS, so it is highly likely that all 6.033 clients already trust it.

2.7.3 Graceful Kill

The ability to gracefully kill an HTTP(S) connection makes the protocol an ideal choice for transferring files. HTTP(S) connections are implemented on top of TCP and have two channels: one for receiving data and one for sending data. In order to close an HTTP(S) connection at any time, one of the endpoints can close its sending channel. If the other endpoint is responsive over the network at this point, they will notice this and automatically close their own sending channel, thus severing the connection. In the case where the other endpoint is unresponsive, after a certain amount of time during which no data is sent or received (about 30 seconds for many HTTP implementations), each endpoint will assume that the transfer has failed and the connection will automatically be closed. The client can then take the initiative to restart the transfer as soon as a reliable route to the server is reestablished.

2.8 Gradescope

In order to maintain records of assignments submitted through Gradescope, the MUGS Server performs a sync operation using Gradescope's API at the same time every day. If there has been a change in Gradescope's set of grades in the last 24 hours, MUGS pulls the set of all Gradescope grades, the Gradescope-generated list of all grades that have changed since the time of the last grade pull, and modifies any changed grades files to reflect the grades and comments provided by Gradescope. To input both new and regraded grades from Gradescope, the system immediately writes the values in the pulled csv to appropriate students' assignment grade files via a procedure identical to staff grading (described in Section 3.3.4).

Though pulling grades from Gradescope takes less than a second in the average case (as detailed in Section 4.3.2), we believe that a propagation time of at most 24 hours between

Gradescope and MUGS is appropriate. There are only 9 assignments in 6.033 that are ever graded through Gradescope throughout the entire semester, and regrades are unlikely to be frequent or time-sensitive enough to warrant a more frequent update period in the MUGS portal. In addition, we believe that a 24 hour delay is widely accepted by students, prevents students from frequently checking their grades (and increasing system load), and is an acceptable trade-off for the added simplicity of only synchronizing once per day.

3. Use Cases

3.1 Students

Aside from uploading files (whose process is described in section 2.6.1), we anticipate that 6.033 students will use MUGS to submit assignments, view grades, and vote for videos.

3.1.1 Submissions

Students have directories associated with each assignment, such as `/6.033/kerb/student_1/assignments/critique1`, that contain assignment instructions and `files` and `submissions` subdirectories. Students may upload intermediate work into `files`. When a student wishes to submit a file, they are presented with a page in the MUGS front-end that lists their stored files. A student can mark a file in `files` for submission by selecting it in a list of files in `/6.033/kerb/student_1/files` displayed on the MUGS front end; MUGS then creates a symbolic link in `submissions` to the selected file in `files`. A student can submit and resubmit assignments as many times as they wish; we maintain a history of all submissions as described in Section 2.4.2.1.

3.1.2 Viewing Grades

Staff upload grades and comments into a separate folder (e.g. `/6.033/kerb/ta_1/critique1/student_1/`). Each student assignment directory also contains a `grades` folder. Once grade release is approved by staff as detailed in Section 3.3.5, MUGS creates a link between the TA's grade file and the student's `grades` folder so students can view their grades.

3.1.3 Voting for Videos

Our design accommodates special assignments in which students submit ballots ranking videos created by other project teams. When students submit ballots, the system can screen ballots for possible voting abuses (such as voting for the same team twice, submitting multiple ballots, etc.) by running an assignment validation bash script specified in the video assignment description. For example, students can submit ballots through the MUGS front-end by choosing teams to assign rank 5 (best) to 1 (worst). The front-end can send this data to the MUGS server, which

processes that ballot into a spreadsheet of format “rank number : team number”. During this processing, the server can also check if the student has voted for the same team twice, and send a request to the MFS to check if the student has already submitted a ballot (and, if so, subtract the student’s previously tallied votes).

Vote totals can be summed in a file in an appropriate staff member’s folder (such as `/6.033/kerb/lecturer/files`) as part of processing. A potential way of doing so is to keep a spreadsheet of “totals : team number”, and to sum the ranks each student submits for each team (leaving teams that the student did not vote for unchanged) when processing a student’s ballot. This allows easy identification of the winning team, which will have the highest total.

3.2 Teams

One of 6.033’s core components is the team-based final design project which students complete in teams of three. Teams must share files, turn in group submissions, be graded as a group, and be able to see and vote on files from other teams. Our design enables reliable, secure, easy-to-use, and flexible implementation of team-based functionalities that are not supported by 6.033’s current grading system.

3.2.1 Submissions and Grades

Since each team has their own team Kerberos, team file uploads and assignment submissions follow the functionality described for students in Section 3.1 using a shared folder like `/teams/teamKerb/assignments/DPPR/`. Similarly, viewing grades and comments as a team closely follows the functionalities described for students, except for a given grade, students now reference a grade file shared among members of the team in a folder like `teamKerb/assignments/DPPR/grades/`.

3.2.2 Forming Teams

Teams must be composed of students attending the same tutorial; thus, team formation naturally takes place in a student’s tutorial folder, which is linked to their MUGS home folder. Since all students in the tutorial have access to the tutorial folder, the tutorial TA can use it to coordinate team formation and create team directories as subdirectories of the tutorial directory, setting permissions and creating links to student home directories as necessary.

One possible way that team creation can happen in MUGS is to maintain a student read-only text file containing the names and contact information of unpaired students in `/6.033/kerb/tutorial_n/files/unpaired_students`. Students can create teams by sending a form containing names of their chosen teammates through the front-end. Upon receiving the form, the MUGS server creates a file in `files`, “TutNTeamX”, containing the newly-teamed students’ names; it also removes those students’ names from the list of unpaired students. When all students are paired (as evidenced by an empty unpaired students

document), the tutorial TA can send a request through the front-end to create the team directories under `/kerb` and link `/parent/team` in each student's home directory to their team home directory. Each new team directory is created after checking for the highest-numbered team directory created thus far (ex. team_z), acquiring a write lock on the directory team_z+1, ensuring that it does not yet exist, then creating the new directory and releasing the lock. If the directory is found to exist after acquiring the lock, that means another TA was creating folders concurrently, so the process must be retried.

We believe that social etiquette (and tutorial instructors' abilities to undo team formation by adding names back to the list of unpaired students) will prevent students from forming teams with students that they have not previously communicated with, and that by providing students with contact information, any stragglers can form teams without needing TA intervention. Thus, this procedure's autonomy enhances user experience simplicity and is general enough to scale to arbitrary groupings in other classes.

3.2.3 Coordinating Files

Team assignment directories (e.g. `/6.033/kerb/teamKerb/assignments/DPPR/`) are accessible by all team members, who can use the directory's files to collaborate on assignments. For reliability, MUGS uniquely names each uploaded file using the uploader's kerberos, assignment name, and timestamp of submission so that users can not silently overwrite other users' files.

Through naming, the system can keep track of which user uploaded the most recent submission for a given assignment and display that information when students try to submit a given team assignment. Though this approach sacrifices a bit of simplicity on the system's part, it greatly increases reliability as it prevents team members from unknowingly overwriting other members' submissions.

3.2.4 Sharing Work With Other Teams: Videos and Reports

Two team assignments – a report and video – need to be viewed by other teams. We assume that the course lecturer allows assignments to be shared among teams only after all teams have submitted the appropriate assignment X to `/6.033/kerb/teamN/assignments/assignmentX/submissions/`; thus, since the lecturer can keeps track of how many teams have submitted a given assignment (via the procedure outlined in Section 3.3.5), they can similarly initiate the sharing process for any specified assignment via the MUGS front-end. Upon form submission, the MUGS server will link the specified assignment submissions (which can be collected through a path such as `/6.033/teams/*/assignments/assignmentX/submissions/*`, then choosing the submission with the highest timestamp for each team) to an appropriate read-only folder within `/6.033/shared` (such as `/6.033/shared/assignmentX`), which everyone can access.

3.3 Staff

Course staff are responsible for setting up the file system structure, grading student submissions, publishing grades for assignments, and monitoring each student's performance.

3.3.1 Creating Students

We assume that the course lecturer has a spreadsheet of all the students taking the course at the start of the semester. To create students, the course lecturer can request (through the front-end) to run a script creating a home directory for each student under `/6.033/kerb`. At the beginning of the term, links to a student's recitation, tutorial, and team in `/6.033/kerb/studentKerb/parents` point to `/6.033/recitations/unassigned`, `/6.033/tutorials/unassigned`, and `/6.033/teams/unassigned`, respectively.

To minimize staff workload and increase user-friendliness, students choose their tutorial and recitation on a first-come, first-served manner through the MUGS front-end. This necessitates some manual work in resolving conflicts (e.g. if a student can only attend a tutorial or recitation that has become full), but we believe that such cases are relatively rare, and that manual conflict resolution is the most effective way to handle such a situation. Additionally, this strategy has been employed in other classes such as 6.UAT with minimal complaints from students. When a student submits the form choosing their recitation and tutorial, the MUGS server runs a script that reassigns the student's parent recitation and tutorial from `/unassigned` to their chosen sections.

If a student drops the course late in the term, we believe that there is no need to remove their access to their team's folder in a time-sensitive manner. At drop date, the course lecturer can take the list of dropped students and run a script (again through the MUGS front-end) that changes the parent of the students' home directories to `/6.033/students/dropped` and creates corresponding links in `/6.033/students/dropped` for each student.

3.3.2 Creating Staff

We also assume that the course lecturer has a spreadsheet detailing the names and roles of each staff member at the beginning of the term, and that this spreadsheet is static throughout the semester (such that the system does not need to automatically handle creating and deleting new staff members). At the beginning of the term, the course lecturer can create a file containing all staff roles in the class; each term in the file will be added as a subdirectory of `/6.033/staff`. They can then initiate a staff category folder population process, wherein each staff member and recitation team gets a home directory (`/6.033/kerb/staffKerb` and `/6.033/kerb/rt_n`, respectively) that is linked to its appropriate parent directories; for example, links are created from from `6.033/rts/rt_n` and other appropriate directories shown in Figure 3 to the staff member's home directory. We believe that this automated

approach to creating staff groupings and home directories increases MUGS' ease-of-use and flexibility.

3.3.3 Creating Assignments

To create an assignment, the course lecturer can input an assignment name and details into a form on the MUGS front end. Upon submission, the MUGS server will add a file in `6.033/assignments` with the description of the file, and create the directories `/students/*/assignments/assignmentN` and `/staff/*/assignments/assignmentN`.

3.3.4 Assigning Grades

For reliability, our file system maintains separate grade files for each party responsible for grading or commenting on an assignment. Each party simply submits their evaluation to MUGS through the MUGS front end, which we envision as having a dedicated grading page displaying the name of the assignment and a space to enter the grade.Then, our system creates the appropriate grade file without any risk of overwriting grades given by other parties. For example, upon submission of a grade through the MUGS front end, the MUGS server can create a text file containing the name of the assignment and the numerical grade in `/kerb/ta_1/assignments/DPPR/student_1`, which is linked to by `/kerb/student_1/grades/DPPR_TA_grade`. Our system only keeps the latest uploaded grade by a given staff member. Late penalties are assigned according to a user-uploaded Bash script as described in Section 2.4.3.

3.3.5 Publicizing Grades

Staff-uploaded grades are initially inaccessible to students. Through the MUGS front-end, the course instructor is provided with a publication function which links all grade files to their appropriate student assignment directories while ensuring that each student has read-only access to their grade files. The system then sends an email to all students notifying them that their grades have been released. The course instructor knows every assignment is graded through a submission counter in the course lecturer's `/submission_counts` directory. Each assignment is associated with a separate count file in `/submisison_counts`, which contains the count of how many unique students have uploaded the assignment and is displayed on the course lecturer's MUGS user interface. Thus, the course lecturer can see when all students have submitted a given assignment and initiate the publication process accordingly.

3.3.6 Viewing and Collecting Grades

Staff can also generate a report of individual student grades in the form of a spreadsheet. This report can be filtered in a few different ways, all of which are easily supported by our file system through the use of the wildcard (`*`) search symbol. The MUGS front-end will have a dedicated

page where staff can view and collect student grades for individual assignments by typing in which student grouping and assignment they want, which the MUGS back-end translates into a search path in the file system. For example, if a member of the recitation 1 staff team wishes to collect all grades for critique 1, the MUGS back-end searches for all grades matching path `/6.033/kerb/rec_1/students/*/grades/critique1`. The course instructor can similarly collect all students' grades for all assignments by the system searching the path `/6.033/students/*/grades/*`. The MUGS back-end aggregates the content of the collected grade files into a spreadsheet mapping student Kerberos and assignment name to numerical grade, which is returned to the user through the MUGS front-end.

4. Evaluation

4.1 Security Model

We believe that our system is reasonably well-protected against students looking to maliciously change grades or access other students' data. All actors accessing the system have to authenticate themselves through MIDS, and our permissions service prevents students from modifying from their grades and from viewing the work of any other user. In addition, students do not have direct access to the MUGS server – they can can only access it through a web interface.

Depending on how the web interface is implemented, it could be vulnerable to a variety of web-based attacks. For example, because we use HTTPS to upload and download files, a man in the middle should not be able to snoop on data during transfer, but a user who can somehow use cookies to impersonate a staff member on the web interface can potentially compromise the system.

We trust that the MUGS server itself or other people who may be using the same machine are not malicious. In general, our system is vulnerable to insider attacks (if course staff become malicious) or against students who can somehow authenticate themselves as staff (and act with staff privileges). If authentication is implemented correctly, we think this should be very unlikely, as that would require a student to know a staff member's Kerberos username and password, or have a copy of their certificate, and MIT also uses Duo to further authenticate Kerberos login.

4.2 Storage Performance

Our system's storage can easily accommodate the needs of a typical class. Assuming each page of a PDF takes up approximately 100 KB and each student or team uploads at most 10 versions of a given assignment, the memory needed by a typical semester of 6.033 is (400 students * 100 KB/page * (10 pages/DPPR + 3 pages/critique + 15 pages/DPR + 1 page/reading question * 26 recitations) * 10 versions) + (~140 teams * (100 MB/video +

100KB/page*(10 pages/DPPR +20 pages/DPR))*(10 versions) = 165,800,000 KB, which rounds up to 166 GB. Thus, our system can be easily accommodated by the provided 240 GB of storage.

Since rich media is so resource intensive, what additional types of rich media our system can support depends on what type of media it is. Audio feedback on video presentations, assuming a 32 Kb/s bitrate and 5 minute length, would take 240 KB/minute * 5 minutes * 140 teams = 168 MB of additional space, which our system can handle. It can not handle, however, another 100 MB video per team because the additional storage that would require, 140 GB, would surpass the allotted storage space for our system.

Although we do not place any hard restrictions on the amount of data that a user can upload into their home directory, we believe this analysis is accurate because course staff can announce expectations for reasonable use of the MUGS filesystem (e.g. for storage of only 6.033-related files), and since MUGS users are not anonymous, any student who abuses the system can be tracked down and punished by the course staff.

4.3 Throughput and Latency Performance

The amount of throughput and latency MUGS can provide is an important metric for reliability. The area where we expect this to matter the most is when students upload files through MSS, especially before a deadline when traffic is high and uploads must happen quickly.

4.3.1 General file upload and download

In the worst case, every team could be uploading their design presentation video to MUGS in the few minutes immediately before the deadline for that assignment. In this case we believe the MUGS server write speed of 6 Gb/s will be the bottleneck in the transfer. We assume that at least 90% of disk writes will be allocated to MSS during such an event, totaling at least 5.4 Gb/s. Since there are there is 1 video per team and about 3 students per team, a class size of 400 students means 133 videos will be uploaded to MUGS at the same time. A 5.4 Gb/s disk write speed can accommodate 133 concurrent writes at a rate of about 40 Mb/s per write. Every student's laptop is assumed to have a connection of 500 Mb/s to the MUGS server, so this rate is easily supported by the students' networks. Thus, if one student from every team uploads their 100 MB video at the same time at 40 Mb/s, it should take all of them about 27 seconds to finish their upload. We believe this delay to be acceptable, even when a student uploads files a few minutes before the deadline.

In the average case, students will be uploading PDFs (each at about 1 MB), not video files, and we predict that it will be extremely unlikely for more than 50% of the class to be uploading files at the same time. As before, if we assume MSS has access to at least 5.4 Gb/s of disk write speed, writing 200 concurrent files to disk gives each file a write rate of about 27 Mb/s. A network speed of 27 Mb/s is supported by all networks involved, so each student's transfer

would finish in about 0.3 seconds. Thus, the time to upload a file in the average case is very short, which should be acceptable by users even under significant time pressure.

Since HTTPS is implemented on top of TCP, students uploading files may experience a slow down introduced by TCP congestion control if there is significant traffic between the MUGS server and the students. This may increase the upload time for files, but will not drop any files unless the network is significantly congested. Since the maximum file size we expect is around 100 MB per upload, it is extremely unlikely that network congestion will be so severe that TCP is unable to maintain a connection between the students and the MUGS server.

4.3.2 Gradescope

In 6.033, Gradescope is used to grade 2 quizzes and 7 hands-on assignments. Assuming each spreadsheet pulled from Gradescope has minimal character encoding overhead, each .csv file will be at most 1 byte/character * (3 characters/grade + 500 characters/assignment name + 20 characters/student name + 4 commas per entry) * 400 students = 210,800 bytes or around 211 KB. We anticipate additional delays from locking grade files to be minimal, as staff are unlikely to manually write to grade files pulled from Gradescope, students can not request to view a grade until it has been published, and students are unlikely to repeatedly view a grade file when checking if an assignment has been regraded.

Thus, in the average, low-use case (assuming Gradescope is refreshing at a time when students impose minimal load on the system), the time taken to transfer grades from Gradescope to MUGS is the limiting factor, and should take 211 KB / (10 Gb/s) < 1 second. In the worst case, Gradescope grades are being fetched while all teams are submitting videos, and while a network brown-out is affecting the MUGS server. In this case, the network speed could be as slow as 1 Kb/s for up to 30 minutes, and since the Gradescope API request has to share bandwidth with the uploading videos, it will take up to 30 minutes to receive all of Gradescope's data. Since the system synchronizes with Gradescope every 24 hours, as explained in Section 2.8, this delay is acceptable.

4.4 Performance Limits on Scalability

The number of additional students and classes our current implementation of MUGS can accommodate is constrained by its currently allocated storage and bandwidth. For example, according to the analysis in 7.5.1, our system can accommodate up to around 580 students given a class with roughly the same assignment requirements as 6.033, but not two classes requiring the same amount of memory as 6.033. However, since video takes up most of the storage in the current 6.033 file system, additional classes that do not require rich media submissions can be accomodated. Since each non rich-media class takes up roughly 25 GB of space, our system as it stands can handle (240 available GB - 166 GB used by 6.033) / 25 GB = 2 such classes, and can not handle being used by every EECS class simultaneously.

As described in Section 4.5.2, 133 concurrent video submissions can be received by the MUGS server at a rate of 40 Mb/s, resulting in an overall length of 27 seconds for all transfers to be completed in parallel. If the number of students were to double in size to 800, then 266 concurrent video submissions would take about a minute, which we believe is the upper limit for what a quick upload should be. Therefore, bandwidth does limit the scalability of MUGS, but not as much as storage space.

4.5 Performance Impacts on System Usability

Here we examine some of the common operations in MUGS to see how long each operation takes under different scenarios.

4.5.1 Collecting Student Grades

As described in section 3.3.6, the course instructor has the ability to generate a spreadsheet of all grades for every student. In 6.033, there are about 20 total graded assignments for 400 students, meaning that there are about 8,000 grade files that need to be parsed to generate the final spreadsheet. If we assume that each grade file is about 1 KB, this corresponds to 8 MB of data to read from MFS. With a read speed of 6 Gb/s, all of the data can be read into RAM in about 0.01 seconds. It will then take at most a few seconds to extract the needed information from each file and compile it into a spreadsheet, resulting in a total time that is well under 10 seconds. Since this is an operation that will be ran a very limited number of times at the end of each semester, we believe this is a perfectly acceptable amount of time for this operation.

4.5.2 Terminating File Transfer

When users transfer files through MSS, for various reasons they may decide to terminate a transfer early. Supporting quick termination directly increases the usability of our system. As explained in Section 2.7.3, HTTPS has a well-defined early termination procedure. With a responsive network, termination happens within two round trip times, and with an unresponsive network, the connection will timeout after not being active. It is common for HTTP to have a connection timeout of 30 seconds, so while a transfer is killed instantly by one endpoint, it may take up to 30 seconds for the other endpoint to be notified of this. Since it is very rare that the MUGS server will be completely unreachable by the student, we believe this is an acceptable amount of time to terminate file transfers in our system.

4.5.3 Creating Student Accounts

According to manual measurements on an Athena cluster computer, it takes roughly .03 seconds to create a directory and 0.04 seconds to create a symlink within that directory. Creating each student's account entails creating four directories: `6.033/kerb/student_1`, `6.033/kerb/student_1/parents`, `6.033/kerb/student_1/grades`, and `6.033/kerb/student_1/assignments`. It also entails creating seven symlinks: links to

team, rec, tutorial links under student's `parents` that point to `unassigned` directories and corresponding links pointing from the `unassigned` directories back to the student's folder, and a link to the student's home directory from `/6.033/students`. Thus, 400 students * ((4 directories * .03 seconds/directory) + (6 symlinks * .04 seconds/symlink)) = 144 seconds, or around 2.4 minutes to create home directories for all students. Given that this operation happens only once a semester, is not subject to time pressure, and is one of the largest-scale filesystem operations that MUGS needs to support, we believe that waiting a couple of minutes to complete it is acceptable.

5. Conclusion

MUGS uses Kerberos IDs and a corresponding hierarchical file system to enable 6.033 students and staff to collaborate on, submit, and grade a variety of assignments. MUGS is designed to prioritize four hierarchical design goals: reliability, security, simplicity of use, and flexibility. Our system prioritizes reliability simply tracking file history through uniquely-named copies of all uploaded files, and utilizing HTTPS to transmit files over unreliable networks. Assigning abstract users to real-world analogues (staff members, students, and teams) and directories to real-world groupings (recitations, tutorials, etc.) ensures security through setting file and directory permissions granted to different users to reflect real-world privileges and flexibility through defining different types of users, groups, and assignments. We achieve simplicity of use through MUGS front-end pages dedicated to different tasks users may wish to perform, as well as automation of bulk procedures. Enabling such simplicity of use may decrease the flexibility of the system to scale to different types of classes, as different functions may be needed to automate different use cases. In addition, prioritizing reliability over performance may impact scalability in terms of class size (and the number of files that can be uploaded for a class). We believe that our design can handle most standard class sizes and file management needs, though future work may focus on improving the flexibility of the system to handle a wider range of class types and sizes.

6. Author Contributions

Design decisions were made with input from both authors, and the report was written and edited by both authors. Jason contributed more to the System Overview section, Shannon contributed more to the Use Cases section, and both contributed to the Evaluation section.

7. Acknowledgements and References

This report was also made possible through the efforts of Kate Nelson, who contributed to early drafts, and Olivia Brode-Roger and Deepti Dhir, who provided technical and writing feedback.