2019 6.033 Design Project:

The MIT Unified Submission and Grading System (MUGS)

See also DP FAQ and DP Errata

Due Dates and Deliverables

There are four deliverables for this design project:

1) **DP Preliminary Report (DPPR)**: This preliminary report will lay out your key design decisions, including both a functional system design and a sketch of any data structures, storage management, and/or network protocols required to achieve your design. It will not include any significant evaluation. It will be approximately 2,500 words and is due March 22, 2019 at 5:00pm.
2) **DP Presentation**: This presentation will address the feedback received on the DPPR, and any corrections or updates to the design project specification. It will also outline evaluation criteria and use cases you will use later for evaluating your design. The presentation will occur during the week of April 17-23.
3) **DP Report (DPR)**: This will be your full report. It will include your final design, all diagrams appropriate for that, your evaluation of your design and a review of how effectively your design addresses the specified use cases. It will be approximately 6,000 words and is due May 6, 2019 at 11:59pm.
4) **Peer Review**: In Tutorial your team will have done an early “review,” providing informal feedback to another team on their design. For this peer review, you will individually review a few specific sections of that (same) other team’s final report, and address some specific questions about that report. It will be approximately 250 words and is due May 10, 2019 at 5:00pm.

Your assignment for each of the four parts above will be distributed in separate “assignment” documents.

The preliminary report, final report, and peer review should be submitted via the submission site on the 6.033 website. As with real-life system designs, the 6.033 design project is under-specified, and it is your job to complete the specification in a sensible way given the stated requirements of the project. As with designs in practice, the specifications often need some adjustment as the design is fleshed out. Moreover, requirements will likely be added or modified as time goes on. We recommend that you start early so that you can evolve your design over time. A good design is likely to take more than just a few days to develop. A good design will avoid unnecessary complexity and be as modular as possible, enabling it to evolve with changing requirements.

Large systems are never built by a single person. Accordingly, you will be working in teams of three for this project. Part of the project is learning how to work productively on a long-term team effort. **All three people on a team must be in the same tutorial.**

Although this is a team project, some of the deliverables have individual components. See the individual assignment links for more information.

Late submission grading policy: If you submit any deliverable late, we will penalize you one letter grade per 48 hours, starting from the time of the deadline. For example, if you submit the report anywhere from 1 minute to 48 hours late, and your report would have otherwise received a grade of A, you will receive a B; if you submitted 49 hours late, you will receive a C.

As stated on the syllabus, you must complete every design project component to pass 6.033.

I. Introduction

6.033 is MIT's required undergraduate class on computer systems design, and MIT is by some measures the world's leading undergraduate institution for teaching computer science. So it is a little awkward that grading in 6.033 currently uses computer systems that are not very well-designed to meet the needs of the class. Your design project for this semester is to design a better submission/grading infrastructure for 6.033, retaining the parts that work well while supplying improvements for the parts that are problematic. You will be designing the back-end of the 6.033 submission and grading system. Others will design and build a user interface for your system, and their requirements constrain your design – but designing that user interface is not your responsibility.

The information system for 6.033 mediates between two broad groups that we'll call students and staff. Both students and staff have capabilities within the class that are partly related to their individual identity and partly related to their membership in one or more groups. For example, students complete some assignments individually but work in small teams on other assignments. Students in one team can see work shared by other team members, but generally can't see the work of other teams. To give another example, much of the teaching staff is divided into teams of one recitation instructor, one TA and one or two WRAP instructors. Staff on a recitation team can see the work and grades of students in their recitations, but can't see the work or grades of students in other recitations.

The current systems used for 6.033 don't support students or staff very well. Here are some of today's problems that we'd like to solve:

1) Grades are not presented to students or staff in any kind of unified way; instead, grades are split across (at least) two different systems.
2) The submission site (used for assignments such as the critiques and design project reports) has a confusing interface with poor feedback. A common error is for a student to submit something like a Google Docs exception webpage rather than the document itself.
3) Although that submission site is used for the submission of both the preliminary and final design project reports, it does not understand teams or submissions by a team. This deficiency means that group members must make difficult choices about whether to submit their own copy vs. trusting someone else to submit on their behalf. Correspondingly, a recitation instructor or TA collecting group reports must engage in a manual process to determine both (a) whether every group has turned in a report and (b) which of multiple submitted reports is the last one for a group.
4) There is little or no system support for forming design groups. Groups must be composed of people who attend the same tutorial, but that constraint is only enforced manually.
5) There is no consistent system for sharing work in progress. The homework submission site is a place where final reports are turned in, but each group must develop its own approach to sharing information among team members or with staff.
6) Deadlines and grading penalties are handled manually. Grading always has deadlines with differing consequences for being late. Some submissions simply do not count if they are late. For others there is an increasing penalty on the grade. One merit of the current system is that it readily accommodates illnesses and other extenuating circumstances.

However, not everything is terrible. Today's systems work well at ensuring that only the "right" people can see submitted work or grades, and the teaching staff likes using Gradescope for grading quizzes. Those good features have to be retained in a "new and improved" system.

The rest of this document provides more information. First, we sketch the people of 6.033 and the flow of submission/grading work that happens in the class. Next, we describe some of the components that you can incorporate into your design. After that, we list what you need to design and how we will evaluate it. Finally, we identify some of the design issues you'll need to consider.

II. Background

Let us begin first with who the players are:

1) Students: about 400 registered students, who are assigned first into recitation sections and tutorial sections, and then into DP teams.
2) There is one each of the following: a Course Lecturer, a Head WRAP Instructor, and an Administrative TA. This staffing may change from year to year; indeed, this year there are two Head WRAP Instructors (they are a team), so the headcount of this group is 4.
3) For each pair of recitations (and associated tutorials) there is a Recitation Instructor, a Recitation TA, and one or two WRAP Instructors. This year there are 18 recitation sections, so there are 9 recitation teams. The students in each tutorial are divided into DP teams of 3-4 students, thus there are typically 15-20 DP teams for each recitation team.
4) Undergraduate Graders: Some number of undergraduate graders (typically at least a handful, but no more than ten) are brought in from time to time to help on some of the simpler grading tasks.

With these in mind, the grading elements of the course are:

1) Hands-on: Each individual student submits each one through Gradescope, so there is no submission in MUGS. The team of Undergraduate Graders grades them (via Gradescope). The grades will need to be transferred from Gradescope to your MUGS system.
2) Reading questions: Each individual student submits each one through MUGS. The student's Recitation TA reads them but does not grade them. In addition, the relevant Recitation Instructor may also read them. (Note that although the current course has reading questions, we do not currently treat them this way. However, the staff wants the future system to work this way.)
3) Quizzes: Each individual student takes one, and the completed quizzes are entered into Gradescope by staff using Gradescope-controlled mechanisms – so there is no submission in MUGS. All the technical teaching staff (all staff except WRAP Instructors) grade them using Gradescope. The grades will need to be transferred from Gradescope to your MUGS system.
4) Critiques: Each individual student submits each one through MUGS. The student's WRAP Instructor(s) and Recitation TA both grade them. The WRAP Instructor(s) and Recitation TA take their actions independently.
5) Participation: There is no submission through MUGS. Each student receives a grade decided jointly by the Recitation Instructor and TA, and entered by either one.
6) Design project (DP): The elements of the DP are:
    a) Design Project Preliminary Report (DPPR): Each team submits a single joint document through MUGS. The WRAP Instructor(s) grade(s) it, and the Recitation Instructor comments on it (but does not grade it). The WRAP Instructor(s) and Recitation Instructor take their actions independently.
    b) DP Team Presentation: There is no submission through MUGS. The Recitation Instructor enters a grade via MUGS, based on the oral presentation by the team.
    c) Design Project Report (DPR): each team submits a single joint document through MUGS. The Recitation Instructor grades it.

d) Peer review: Each individual student submits their own review through MUGS. The student's WRAP Instructor(s) grade(s) it. Note that the Peer Review is a case where a student sees a different student team's work and submits comments about it, which is otherwise the sort of activity that only staff do.

In the current system, there is a kind of versioning. Every submission by a student is recorded, so staff can readily access the most recent one, or the last one before a deadline. Students can submit many versions of things like Hands-on, Critiques, and Design Project Reports (but not quizzes!). Notice that students can also choose not to submit on time, but rather incur the penalty of a late submission if they choose. Every submission is a potential candidate for grading. Ordinarily, a staff member selects the last submission before the deadline to be graded, but a student can request a different one to be graded (usually because there was some last-minute problem). You can decide whether to retain this manual selection system as is, or automate some of this process.

Similarly, staff can submit versions of their grades and/or comments, so that students can see the latest. At present, when grades and comments are uploaded, students can immediately see them. For a consistent experience across the class, grades that are ready early are often delayed so that grades from the whole class are available at the same time.

We wish to retain all these features of the current system. In addition, we would like to add a new component in future classes.

In future classes, there will be an additional component of the Design Project: a short video of the team presenting a poster. Each video will be 5 to 8 minutes long, and one will be submitted by each team. A plausible estimate for each video's size is 100 MB. The video presentation will be graded by the WRAP Instructors. In addition, each student can vote for up to 5 other teams' videos, providing rankings from 1 to 5. This voting will not impact the teams' final grades, but there will be prizes for the top three peer-rated videos.

To summarize: your design will retain the good features of the current system, fix the problems of the current system, and accommodate the new short-video element as a new feature.

III. Infrastructure support

We provide you with some elements of infrastructure, to incorporate into your design as you deem appropriate. Your choices of whether and how to use the infrastructure elements should be both explained and justified. You are not required to use all these elements.

There are five services provided: an identity service, a file service, a sync service, a lock service, and an external grading service (Gradescope). The identity service and Gradescope are outside your control; the others are services that you can potentially modify or extend to meet your requirements. Each of the services provides some facilities that are rather nice, but are also somewhat limited. Your task is to assemble them into a system that supports editing of files by both individuals and groups, submission of assignments by both individuals and groups, and the grading of assignments. We describe each service and then include a simple functional interface to it.

The core of your system will be a single, central server. (That's not necessarily how you would build this system in real life, but is a reasonable simplification.) The following resources of the server are available to you:

Processor speed	2.1 GHz clock speed
Max simultaneous processes executing	10 (10 cores)
Cache	11 MB

Main Memory	16 GB
Storage	240 GB (effective) SSD in mirrored configuration. Bandwidth is 6 Gb/s

There are two terms here that you may not have seen before. First, in a mirrored disk system, all data is duplicated exactly on two independent devices. This provides increased reliability and availability of the storage system, at the cost of half the storage capacity. In this case, the storage capacity you will have available will be 240 GB (which will be duplicated on two devices). Second, disk bandwidth is the maximum rate of transfer between the processor and the disk system total, for all processes that may be trying to access the disk system.

Note that this table reflects what your system can use. The actual server is somewhat larger than this, and the operating system consumes some of its resources. The OS actually consumes a varying amount of resources depending on what it's asked to do, but we have simplified its behavior.

The server's operating system provides processes. One of your tasks will be to decide how best to utilize processes to support the services below, in addition to any other activities you design on the server.

You can expect that each student and staff member will have their own laptop or other computer, and all computers are on the campus network. You can assume that all the personal machines have adequate networking capacity for the tasks at hand, but the other capacities (storage and computation) of these machines will be varied and unknown to you.

In terms of the network, for simplicity we assume some uniformity. We assume that the MIT backbone network runs at 100 Gb/s, the connection outside MIT is 10Gb/s, each wired Ethernet link runs at a maximum of 1Gb/s, and each wireless link can support up to 600 Mb/s (running 802.11ac WAVE 2, although that should not matter to you). You can also assume that each edge device such as a laptop, tablet, or smart phone also supports 802.11ac, which supports at least 500Mb/s. That said, there will be times and locations on the MIT campus where network latency is extremely variable. There are occasional "brown-outs" with the following character: Each second, there is a 0.00058% chance of such a brown-out starting. This corresponds to roughly once every 48 hours, on average. The length of each brown-out is uniformly distributed, with a minimum of 10 seconds and a maximum of 30 minutes. The available bandwidth during each brown-out is also uniformly distributed, with a minimum of 1Kb/s and a maximum of 100 Kb/s. Because of this potential network misbehavior, we anticipate that users will need to be able to stop or kill some kinds of file transfers while they are in progress. Part of your design will be a network protocol that supports this feature.

A note on interface descriptions:

First, as a reminder, although we are providing apparent function calls, this project does not include any code. They are provided to give you a more specific view of the functionality provided by the services. You should not consider submitting code in any of your reporting.

Second, in the interest of making this presentation shorter and clearer, we have omitted some aspects that are repeated everywhere. For every function we describe, you should assume that the function also supports access control as follows:

1) Each function call must include a Kerberos ID (see next section) representing on whose behalf the function call should be executed.
2) Each function can signal "permission denied" if that on-whose-behalf Kerberos ID does not have adequate permission for the function.

In addition, the functions may signal a failure if they fail to complete for some other reason. It is up to you to decide how to respond to these signals.

(Our description of the interfaces in terms of “signals” is simply intended to capture distinctions between normal processing and exceptional or error situations. We are not mandating any particular implementation language or error-handling system, and your design should not depend on using a particular language.)

1. Identity service (MIDS)

The MIT ID service (MIDS) is based on Kerberos, which provides every individual who has an “account” at MIT a unique ID and an approach to trustworthy “logging in” to verify their identity and have privileges associated with that identity. Effectively, MIDS hides all of the messy details of login and authentication, so that your design can focus on what happens with authenticated identities. That said, you need to know that Kerberos_IDs are unforgeable, authenticatable tokens; associated with each one is a Kerberos name. Currently, at MIT each member of the community has a Kerberos name, which is also used as their MIT email address. Each person also has an unforgeable, authenticatable token to use for a variety of functions on campus.

MIDS is an “enhanced” version of Kerberos based on the needs of MUGS, and provides the ability to create new Kerberos IDs: these additional Kerberos IDs may be useful in supporting the groups and teams that are used in 6.033. It is important to note that these new IDs are just IDs, like the IDs of “ordinary” Kerberos. Permission to create these new IDs in Kerberos is restricted, but unlike the creation of new “individuals” at MIT, permission to create group IDs can be delegated. The original permission for the class is delegated to the Course Lecturer, who in turn can delegate to others. Thus, for example the Administrative TA might be able to create recitation section IDs and each Recitation TA might be able to create the IDs for DP teams. Kerberos provides no directory or other lookup service; to use these new IDs effectively for groups and teams, your design will need to include additional data structures and/or services.

That all said, it is important to remember that Kerberos does not keep track of personal information about people, such as names or roles they might play (such as student, staff member, etc.). It also has no way of keeping track of who might be “sharing” an identity, such as one created for a group.

The MIDS IDs are used by the file system described below for access control.

The functional interfaces to MIDS include:

Command	**Action**
create_id () returns (new_kerb_id, kerb_name, new_home-dir)	Creates a new ID, returns the ID and home directory for it
delegate_kerb_creation (kerb_name)	Delegates the ability to create new Kerberos IDs
authenticate (kerb_id) returns (kerb_name)	Authenticates a Kerberos ID and returns the matching Kerberos name.

(Remember the “note on interface definitions” earlier in Section III.)

2. File service (MFS)

The MIT File System (MFS) is a shared file system that provides reliable, stable, sharable files in a hierarchical file system. The file service has typical Unix (open, read, write, close, etc.) interfaces. Our “enhanced” file system supports a richer access control model than standard Unix, because any number of “users” can be given distinct access permissions for each directory or file. The “users” in MFS are Kerberos IDs from MIDS, including those for both individuals and groups. Notice that an individual can have their one individual ID as well as one or more group IDs; for example, each student will be in a group of all students, in a recitation, in a tutorial and in a DP team, and therefore have access to the IDs for all three, in addition to their own individual IDs. This means that when someone wants to execute an operation, they can act using their own individual identity or any group identity of which they are a member. Different identities will have different privileges, so they will need to be able to choose which identity to use for any specific activity.

In addition, MFS offers the capability to *snapshot* a file or directory. The result of a snapshot is an opaque snapshotID data structure. By supplying an appropriate snapshotID to a *revert* operation, it's possible to cause the file or directory to return to its exact state at the point of the original snapshot. It's pretty easy to understand how snapshot and revert work for a single file, or for a directory containing only simple files. Snapshot and revert will also work for more complex situations, but it becomes increasingly difficult to be sure what the correct or expected behavior is. Your system should avoid relying on knowing exactly how MFS works for such edge cases.

A snapshotID can be tested for equality or tested for order (newer vs. older). A snapshotID can also be tested for whether it was generated by a particular file or directory, but does not otherwise reveal any aspect of the snapshot implementation.

Concurrent operations on any file or directory happen in an undefined order, and are not guaranteed to be atomic. This lack of ordering guarantees applies to both conventional file operations and snapshots.

Each Kerberos ID that is being used in your MUGS system has a corresponding home directory in the filesystem. In addition, the Kerberos IDs are used as the basis for setting access control on files in the filesystem, including control over who can change the access control on a file or directory. Access control is at the whole-file or whole-directory level only, not at any finer grain.

We underscore that even if a Kerberos ID from MIDS is intended to represent some kind of team, it still looks like a "user" to MFS – not a Unix "group."

The functional interfaces for MFS include:

Command	**Action**
create_file(filename)	Creates the specified file with permissions as specified by the parent directory
read_file (filename) returns (content)	Reads the specified file
write_file (filename, content)	Writes the content to the specified file
create_dir (directory name)	Creates the specified directory with permissions as specified by the parent directory
create_snapshot (path) returns (snapshot)	Takes a snapshot of a file or directory
revert_to_snapshot (snapshot)	Reverts file or directory to a previous snapshot
path_from_snapshot(snapshot) returns(path)	Identifies file or directory captured by this snapshot
snapshot_eq (snapshot1, snapshot2) returns (bool)	Tests for equality of snapshots, returns true if equal, false otherwise
snapshot_earlier (snapshot1, snapshot2) returns (bool)	Tests whether snapshot1 is earlier than snapshot2, returns true if snapshot 1 is earlier

change_access_control (kerb_id, filename, permissions)	Changes the permissions for the named file or directory to include the specified permissions for the specified Kerberos ID

(Remember the "note on interface definitions" earlier in Section III.)

3. Sync service (MSS)

The MIT Sync Service (MSS) supports connections from a client (typically the laptop of a student or staff member) to a server (typically MFS). For any file on the server that is accessible to the client, the client can *download* the file, getting both a local copy of the file and an associated (but hidden from you) tag. The hidden tag of a file consists of some kind of timestamp, version number, or checksum – its precise implementation is not important for this project, and your project should not depend on the characteristics of any particular implementation approach.

The client can modify the local copy of the file using any of their usual local ways of modifying a file, and can subsequently attempt to *upload* the file via MSS. The upload will replace the server's copy of the file with the client's version if and only if the server's copy is unmodified since the client's download. If the server's copy has been modified, the attempt to upload will fail. You will need to decide what (if any) action is taken by the system under this condition.

The sync service hides the details of the data encoding. You can assume that a scheme like delta encoding is in use, so a small change to a large previously-transmitted file will only result in a small amount of data crossing the network in either direction.

Concurrent uploads on the same file happen in an undefined order, but each file's upload either succeeds completely or fails cleanly. Concurrent uploads on multiple files in the same directory happen in an undefined order and are not guaranteed to be atomic: that is, some files may upload while others fail.

MSS has one additional feature: the client can kill an upload or download while it's in progress. This is most likely to be used if the transfer is taking too long. As mentioned earlier, the network experiences wide swings in latency. Killing a transfer produces the same effect as a clean failure: there is no change to the file(s) involved in the transfer.

The functional interfaces for MSS include:

Command	**Action**
sync_download (filename) returns (local_filename)	Downloads a copy of the file to a local filename
sync_upload (filename, local_filename)	Uploads a copy of the local file to the remote filename
kill (filename)	Sends a kill message for a transfer for the given filename between the client and server.

(Remember the "note on interface definitions" earlier in Section III.)

We will ask you to design part of the network protocol that supports transferring data between client and server. A different part of the implementation of upload/download (not designed by you) will determine what bytes need to be transferred – some transformation of the bytes in the file being transferred. Your protocol will be handed these (transformed) bytes and a file identifier recognized by the transformation machinery. On the other side of the network, you will hand off the complete collection of bytes (in the same order) and the file identifier; or you will indicate the transfer has been killed.

The total number of bytes to be transferred will often be larger than a single Ethernet packet, so you will need to find a way to transfer a large quantity of bytes while preserving their order. A TCP connection would allow you to transfer a large number of bytes in order, but TCP does not have a well-defined capability to cleanly kill a partial transfer. You will design some solution that supports both the ordered delivery of a large number of bytes, and also allows for a clean "kill" of a partial transfer.

4. Lock service (MLS)

The MIT Lock Service (MLS) provides a single shared namespace of locks. A lock is a mechanism for coordinating concurrent activities, by ensuring controlled access to some shared resource. Each lock can be simple (exclusive) or reader/writer.

If two clients use the same name (text string) in referring to a lock, they are referring to the same lock. If a lock is currently held, the lock service will report the Kerberos ID(s) of the lock holder(s).

If two or more clients use the same lock in different ways (one uses as exclusive, another as reader/writer) then the lock's behavior is undefined. You will need to address this in your design.

The functional interfaces for MLS include:

Command	**Action**
Acquire_exclusive (lock_name) signals (held(kerb_id))	Acquires the specified lock, excluding all others until release. May signal with ID of holder if lock is held.
Acquire_writer(lock_name) signals(held_writer(kerb_id), held_reader(set[kerb_id]))	Acquires the specified lock, excluding all others until release. May signal held_writer if another ID holds lock as writer. May signal held_reader if one or more other IDs hold lock as reader.
Acquire_reader(lock_name) signals (held_writer(kerb_id))	Acquires the specified lock, allowing other readers. Would-be writers are excluded until all readers have released. May signal held_writer if another ID holds lock as writer.
Release_lock (lock_name)	Releases the named lock

(Remember the "note on interface definitions" earlier in Section III.)

5. Gradescope

Gradescope is an online grading facility. 6.033 uses Gradescope for grading all the hands-on submissions and both quizzes.

Items to be graded are entered into Gradescope by mechanisms outside the scope of this project. Grading (and regrading!) are likewise handled by Gradescope's interfaces and mechanisms, and we won't consider those issues further. However, you will need to extract grades from Gradescope into your MUGS system. Our Gradescope interface for this design problem is fictional, but derived from the real interface.

For exporting grades, our Gradescope provides an export function, which returns a CSV (comma separated values) file containing a snapshot of all the grades available at that time for all students for all assignments for that course being handled by Gradescope. The file that is returned is static and is not updated with any additional or changed grades in Gradescope. Successive calls to the export function may produce different sets of grades; each one is a snapshot of the grades at that time.

Gradescope is configured to use student email addresses (@mit.edu) to identify students. This means that the exported CSV file will have text names that can be easily converted to Kerberos IDs or vice-versa; however, it is important to keep in mind that these names in Gradescope are just pieces of text. Gradescope does not understand Kerberos IDs in the same way that MIDS and MFS do.

Because it is useful to know when grades change or new grades are added (for example when students take make-up quizzes), Gradescope provides a notification function: this function will return only when some grade (relevant to 6.033 grading in Gradescope) is entered or changes. Unfortunately, the only way to figure out exactly what has changed is to export the CSV version of grades again. The export takes longer as there are more grades to report; at the maximum number of grades exported for 6.033, the process takes about 10 seconds. Although it may be useful to be notified when a regrade happens, you likely will not want to be notified and re-exporting the whole CSV file every time a grader enters or changes a single student's grade on a quiz or hands-on problem. There will be an especially high rate of change in Gradescope grades as the quizzes and hands-on problems are graded. At a lower rate, there will also be changes in the grades when regrading occurs.

Gradescope is operated by a startup company in Berkeley, California. Our Gradescope service on the server at MIT makes remote calls to the "real" Gradescope service. The functional interface for our Gradescope service includes:

Command	**Action**
Pull_gradescope_grades() returns (CSV_data_structure)	Acquires a snapshot of the current grades available for the whole course through Gradescope.
Change_in_gradescope_grades (timestamp)	Returns when there has been a change somewhere in the complete set of Gradescope grades for the course, after the time specified by "timestamp"

(Remember the "note on interface definitions" earlier in Section III.)

IV. The Design project

In this section we outline the requirements for your system, as well as the criteria we will use to evaluate your design.

A. Your design

Your design must include the following features.

1) To support student work:

   a) A facility for teams to share individual and team files consistently. Team members can edit files offline independently, so your design must avoid losing updates due to conflicting changes. (If Alice produces a new version of some file and Bob produces a different new version of the exact same file, the file sharing facility must not allow Alice's version to silently overwrite Bob's version, or vice-versa).

   b) A coherent approach to the organization, storage, and accessing of assignments, submissions in response to assignments, and grades and comments on those submissions.. This will need to include a submission function for submitting files as responses to assignments. Note that submitting is distinct from uploading a file into the file system, and correspondingly that submissions are not the same as files. The submission function will allow students individually and in teams to submit their files to be associated with an assignment when they choose. Any submission may be made repeatedly, because students may want to submit more than once before a deadline. The system must keep a full history of submissions. A file needs to be designated in your system as part of a submission to a particular assignment by a particular Kerberos ID (individual or team). As described in Section II, submissions may be either documents or videos.

   c) Categorization of submissions as on-time (before the deadline) or late. Depending on the assignment, there may be several different categories of "late" with different penalties.

2) To support staff work:

   a) A collection function that allows staff to download all student/team submissions for a given assignment.

   b) A grading function that allows staff to upload comments and assign grades. Staff may upload multiple versions of their grades and/or comments, singly or in bulk. In contrast to student submissions, these successive versions are not recorded; only the last version is retained.

   c) The grading function must allow more than one kind of person to submit comments. For the Critiques, the WRAP Instructors and TAs will both be commenting. For the DPPR, the Wrap Instructors and Recitation Instructors will both be commenting.

   d) A publication function that makes staff comments and grades visible for a given assignment. Before publication, no student sees any grade or comment on the assignment. After publication, all available grades and comments are visible to the appropriate student.

   e) Gradescope integration: Hands-on submissions will continue to be handled via Gradescope and not through MUGS directly. Likewise, quizzes will be handled via Gradescope. However, grades for the Hands-on assignments and quizzes will need to be exported from Gradescope and into your system in such a way that they become grades in MUGS.

3) To support grade reporting:

   a) A reporting system that allows recitation teams and the Course Lecturer to appraise the performance of individual students, whole recitation sections, or the whole class. Staff may request such reports for a single assignment or for all assignments.

4) To support the new poster video submissions:

   a) Rank-ordered voting on the videos, one ballot per student, in order to compute how many students voted for each video.

5) To support efficient use of resources:

   a) A sensible allocation of tasks/functionality to processes on the server. The process structure will include both the infrastructure services described in Section III and any additional capabilities you will need to provide the overall MUGS capabilities.

   b) A sensible allocation of storage space on the server, dividing it between uses such as home directories, shared team space, storage of submissions, storage of grades, and any other data your design needs to store.

   c) A protocol design for transport of sync service data, supporting the capability to kill a currently-active sync operation.

Your design choices will be based on correctness of functionality, performance, and flexibility. These are discussed in the following section.

We are not asking you to write any code; indeed, we don't want to read any code. We're asking you to architect a system for others to implement. Your reports will focus on major choices and tradeoffs. You will need to write clearly and precisely, in a way that helps the reader to understand the structure of the system. You will also need to construct diagrams that likewise clarify the overall system structure or particular aspects of its operation.

B. Evaluation Criteria

The system you design must meet a number of criteria that relate to correctness, performance, and flexibility. The general criteria identified below will be supplemented with additional evaluation criteria later in the term.

CORRECTNESS

Your system's operations must allow the necessary flow of information.

1) **File system view**: Because files can be edited offline and some files are being managed jointly by teams, it is important to avoid lost updates due to independent but conflicting edits.
2) **Individual submissions**: Students must be able to submit their individual assignments; relevant staff must be able to see those assignments and grade them.
3) **Team submissions**: Students on the same design team must be able to share files that are part of the team's work. Any student on the design team must be able to submit those team assignments. Relevant staff must be able to see those assignments and grade them. In addition, students must be able to see the DP Report that they will be peer-reviewing and each others' videos.
4) **Student view**: Each student must be able to see their own grades and comments, as well as the grades and comments they have earned as a team member. Each student must be able to see the final DP Report they are assigned to peer review. In addition, every student must be able to view all other student videos.
5) **Staff view**: Each staff member must be able to see all submissions that they are currently grading, as well as any partial grades or comments they may have assigned to those submissions.
6) **Recitation team view**: Each staff member of a recitation team must be able to see all submissions of students and teams in that recitation. Each staff member of a recitation team must be able to see all completed grades or comments that have been assigned.
7) **Course Lecturer view**: The Course Lecturer must be able to see all submissions, grades and comments for any student.

However, your system must also ensure that people can only see what they are supposed to see. In particular, students must not be able to see the individual grades or comments of other students, even on the same team.

Your system must also support close integration with Gradescope, so that MUGS is a reliable source of information about course grades overall. At the same, remember that MUGS will not be providing the detailed interface to Gradescope, but rather students and staff will continue to use Gradescope generally as they do now. So Gradescope submissions, detailed grading, and regrading will all continue to be handled directly in Gradescope. The interface between Gradescope and MUGS will be only in the form of the two functions listed early in this document, for extracting the current full set of grades for all assignments, and the signal of availability of some new grade(s). Thus, as an example, MUGS will not contain the details of grading for each quiz problem for each student, but only the total quiz grade for each student. For the detail on each problem, students will still need to access Gradescope directly.

PERFORMANCE

1) The system must provide rapid and reliable feedback to students during submittal. The system responds to each submission promptly, and the system's response means that the submittal is guaranteed to be stored safely.
2) The system must perform reliably despite spikes in demand. You can expect that 90%+ of final submissions are likely to happen in the last two minutes before a deadline. In addition, at least 50% of submissions are submitted two or more times in the last five minutes before a deadline.
3) As updates occur in Gradescope, your system will sometimes be out of sync with it. Your design should make a sensible choice about how large you can allow this difference to be. You must decide how frequently to poll for new data from Gradescope and defend your decision about the overhead costs incurred.
4) The system must be able to deliver all student grades for all assignments to the Course Instructor in a reasonable amount of time. Less than 10 seconds is certainly acceptable, somewhat longer might be OK.
5) We do not give a specific requirement for how rapidly a file transfer must be killed. However, we will ask you to estimate how rapidly your mechanism will do so. This will probably be influenced primarily by your design of MUGS, although it is possible that the network fluctuations discussed earlier may have an impact on this as well.

FLEXIBILITY

As specified, MUGS is designed only to support 6.033, and mostly deals with text documents and grading. Both of these aspects might well change in the future. Accordingly, your system should be designed with an eye on these two possible future changes:

1) Scaling up to more undergraduate classes: The rest of MIT is watching your success. First Course 6 plans to use MUGS for all Course 6 undergraduate courses, rolling this out in stages, noting that a small number are significantly larger than the 400 students in 6.033. In addition, there will be many more courses to be handled simultaneously. In the longer run, all of MIT is considering using MUGS. As an estimate, if each student is taking 4.5 courses (some are taking 4 and some 5), there are 50 undergraduate courses being taught in each term, and there are, in round numbers, 5,000 undergraduates, consider how your system might need to be extended to handle larger courses and increasingly more courses simultaneously.
2) Rich media: In this design we have started down a path of storing video, but other assignments in the class could also be video or audio submissions. For example, we could make video recordings of design presentations, and then provide audio annotations from the Recitation Instructor as commentary. How would your system accommodate this evolution in stored/processed content?

Note that your system as specified in your report does NOT need to meet either of these requirements; but you will need to be able to explain how you could modify it to meet them in the future.

V. Design tradeoffs and use cases

A. Design Tradeoffs

There are a number of design tradeoffs possible in this project. We highlight some of them here, but there are likely to be more.

1) You will need to organize the system's information, both data and metadata (data about the data). There are many different ways of organizing the relevant information, with different advantages and disadvantages. You will need to decide how to do this and justify your choice.
2) Within the central services, you have choices for how to manage coordination. It is important to notice that each group with access to a directory or file provides an "opportunity" for conflicting modifications. You will need to decide how those conflicts will be handled (Prevented? Mitigated? Ignored?) and explain your solution(s).
3) As described, there are no constraints on the storage or computation that can be used by an individual or team – a situation that seems ripe for abuse. How should the resources of the central services be managed, so as to provide continuing adequate service?
4) You will need to decide which aspects of the course's operation are enforced by your system vs. which aspects are enforced only by convention and common sense, as well as explaining why those are the right choices.
5) Gradescope offers only a limited and slow means of extracting grades in an inconvenient format. How often should you get those grades, and should that export be triggered by events, time, or some combination? How should you maintain Gradescope-derived grades in your system?
6) The "penalties" on late grades raise interesting tradeoffs. Does the system enforce penalties or simply notify the user that the condition exists? As you may know, there are late submission conditions under which penalties may not be applied, such as illness with S3 support, so your system must allow for that in some way, but how that is handled is your choice.

B. Some use cases

Here are some examples of situations that are likely to arise when your system is in use. We provide these to help you think through some of the less-obvious issues in your design, and we will ask you to describe your system's behavior in at least some of these use cases.

Note that you are not in any way limited to considering *only* these situations – you are free (and encouraged!) to consider other use cases as well.

1) **Bulk creation of accounts:** At the beginning of term, accounts need to be set up for roughly 400 students in a short span of time. Staff accounts have most likely been created in advance of that point and the staff accounts arranged into recitation teaching teams. You should be able to explain how your system goes from "no students" to the fully-functioning "all students" situation. Note that this most likely involves more than just creating an account for each student, since the students also need to be arranged into recitations so their later submissions and grading work correctly. As part of this use case, you should be able to explain approximately how long this process takes.

2) **400 simultaneous submissions:** It's deadline time! Everyone has been procrastinating and they are turning in the assignment at pretty much the last possible minute. Does your design demand resources (processes, network, storage) beyond what's available? (What's the worst kind of assignment for your system in this scenario?)

3) **Notification of 400 students simultaneously that a grade was posted**: The grading is all done and now it's time for the students to be notified. What's the state of the system just before publication, who's taking the publication action, and what happens when they do?

4) **Regrade of quiz question for one student in Gradescope**: The quiz was already graded and the grades from Gradescope have been merged into your system (however that happens). But now someone has successfully appealed the grading of one question and their grade has changed in Gradescope. How does your system notice, how long elapses after the Gradescope change, and what happens in your system to incorporate the regrade?

5) **A student drops the course late in the term:** Because MIT allows students to drop the course, you must consider how you will handle the impacts of that. First, your system will have been keeping track of grades for every student. How do you handle the fact that a student may be removed from Gradescope, so the list of grades you retrieve of Gradescope may not be perfectly synchronized with the students represented in MUGS. In addition, if DP teams were already formed, that student will also have been put into a DP team with the access privileges of that team. How will you handle that?

6) **End-of-term spreadsheet for grades meeting:** The final output of 6.033 – as far as the Registrar is concerned, anyway – comes after the final exam has been graded, when all the staff meet to assign final letter grades. For that meeting, the Course Lecturer produces a final report of all the grades, for all students, on all assignments. How does that work in your system? Do you include ALL students, even the ones who've dropped the class?