The Development and Management of GradeSuite: A Microservice LMS for Mastery Learning

Connor Bernard
Dan Garcia, Ed.
Armando Fox, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-127
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-127.html

May 21, 2025

Acknowledgement

Dan Garcia for his invaluable guidance, mentorship, and unwavering support throughout this research project.

The development teams behind GradeView, Concept Map, GradeSync, Instructor Dashboard, and the upcoming AutoRemind project.

Cisco, for the wealth of knowledge that I absorbed during my time working there concurrently as I architected and developed GradeSuite; my experience there was invaluable in guiding the development of GradeSuite's framework.

The Development and Management of *GradeSuite*: A Microservice LMS for Mastery Learning

by

Connor Robert Bernard

A research project submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science (EECS)

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Dan Garcia, Research Advisor
Armando Fox, Second Reader

Spring 2025

The Development and Management of *GradeSuite*: A Microservice LMS for Mastery Learning

Abstract

The Development and Management of *GradeSuite*: A Microservice LMS for Mastery Learning

by

Connor Robert Bernard

Master of Science in Electrical Engineering and Computer Science (EECS)

University of California, Berkeley

Traditional Learning Management Systems (LMSs) often lack the infrastructure necessary to support mastery learning, a pedagogical approach that emphasizes conceptual understanding and flexible assessment timelines over rigid deadlines. This limitation creates significant barriers for educators seeking to implement mastery-based grading policies. To address this gap, we present *GradeSuite*, a microapp-based LMS designed specifically to facilitate mastery learning implementation while integrating with existing educational infrastructure.

GradeSuite comprises four primary microapps: *GradeView*, a sophisticated dashboard pro-

viding granular concept-level feedback; *Concept Map*, offering interactive visualizations of learning progression; *GradeSync*, enabling automated grade synchronization across multiple LMSs; and *Instructor Dashboard*, delivering comprehensive analytics for course staff. This architecture allows *GradeSuite* to aggregate and analyze grade data from diverse sources, providing both students and instructors with detailed insights into conceptual mastery while significantly reducing administrative overhead.

We deployed *GradeSuite* in UC Berkeley's non-majors computer science course — *CS10: The Beauty and Joy of Computing (BJC)* — across two semesters, serving 218 students total. Our results demonstrate substantial improvements in both educational and operational outcomes. Students actively engaged with the platform, averaging 18-19 uses per student per semester, with the majority reporting that it helped them make informed decisions about retaking exams and identifying knowledge gaps. Interviews with course staff revealed that *GradeSuite* reduced grade management time by 90% (from 10 hours to 1 hour weekly) while eliminating final grade calculation errors. The automated grade synchronization provided by *GradeSync* significantly improved grade update frequency and accuracy, addressing a key student concern.

This report details *GradeSuite*'s development, architecture, and deployment strategies, including its evolution from cloud to bare-metal and back to cloud infrastructure. The project's success is supported by a robust organizational structure that grew from an individual initia-

tive to a team of over twenty contributors, managed through a carefully designed hierarchical framework that promoted both accountability and open communication. We also discuss future developments, including *AutoRemind*, an intelligent notification system designed to support flexible deadlines and students continuing work beyond the traditional semester. Our findings suggest that *GradeSuite* provides a scalable and effective solution to support mastery learning.

This report is dedicated to:

my parents, whose unwavering support and encouragement have been the foundation of all my achievements;

my brother Trevor, whose academic excellence and dedication have been a constant source of inspiration, challenging me to push beyond my perceived limits;

and

Shayla Whitely, whose guidance in developing my executive functioning skills has been transformative — the tools and strategies she taught me have been instrumental in my academic journey and made this achievement possible.

Contents

List of Figures

List of Tables

List of Listings

Acknowledgments

I would like to express my deepest gratitude to Professor Dan Garcia for his invaluable guidance, mentorship, and unwavering support throughout this research project. His expertise and insights have been instrumental in shaping both this work and my growth as a researcher.

I am profoundly grateful to the authors of the original *GradeSuite* paper, whose foundational work laid the groundwork for this research. Their contributions have been essential to the development of this project.

This work would not have been possible without the dedication and technical excellence of the development teams behind *GradeView*, *Concept Map*, *GradeSync*, *Instructor Dashboard*, and the upcoming *AutoRemind* project. Their collaborative spirit and commitment to quality have been crucial to the success of this project.

I would also like to extend my appreciation to Cisco for the wealth of knowledge that I absorbed during my time working there concurrently as I architected and developed *GradeSuite*; my experience there was invaluable in guiding the development of *GradeSuite*'s framework.

Finally, I am thankful to the broader academic community at UC Berkeley for fostering an environment of innovation and intellectual curiosity that has enabled this work to grow to what it is today.

Chapter 1

Introduction

The landscape of education is rapidly evolving, yet traditional grading structures remain anchored in assessments and policies that often fail to capture students' conceptual understanding and growth over time. Mastery learning, a pedagogical framework that emphasizes topic-specific proficiency and knowledge demonstration over rigid temporal constraints, offers a promising alternative to these conventional approaches [5]. However, the widespread adoption of mastery learning policies and practices is often limited by Learning Management Systems (LMSs) that do not provide appropriate support and customization.

To address this gap, we present *GradeSuite*, an innovative LMS architected specifically to facilitate the implementation of mastery learning policies. *GradeSuite* substantially reduces administrative overhead while providing more instructor flexibility than conventional LMSs. The platform's distinctive approach lies in its ability to aggregate and visualize grade data from diverse existing LMSs, providing students with granular feedback on their conceptual

mastery, while offering instructors comprehensive insights into course-wide understanding at a conceptual level.

GradeSuite is comprised of a series of microservice-based applications (microapps) including *GradeView*, *Concept Map*, *GradeSync*, and *Instructor Dashboard*, each of which is designed to address specific aspects of mastery learning implementation. *GradeView* represents a paradigm shift in assessment methodology, providing instructors with a sophisticated grading platform, enabling them to transition evaluation from traditional assignment-based metrics — where grades are computed as a sum of grade components — to concept-focused assessment previously unsupported by existing LMSs. This granular approach, supported by infinitely-flexible instructor defined formulas through the familiar Google Sheets platform, enables precise measurement of student understanding across individual topics.

Complementing this, *Concept Map* delivers personalized learning pathways through interactive visualizations of mastery progression, while *GradeSync* ensures seamless real-time data synchronization across multiple LMSs and semesters, facilitating extended learning opportunities beyond traditional term boundaries. The *Instructor Dashboard* component serves as *GradeSuite*'s analytical cornerstone, offering course staff unprecedented visibility into both aggregate and individual student progress. This comprehensive analytics platform provides instructors with insights into student knowledge acquisition that surpass the capabilities of traditional LMSs. The integration of these components creates a cohesive ecosystem that supports both granular assessment and holistic understanding of student progress.

Initial deployment of *GradeSuite* in UC Berkeley's introductory non-majors computer sci-

ence course, *CS10: The Beauty and Joy of Computing (BJC)*, has yielded promising results, with both instructors and students reporting positive experiences and expressing optimism about its potential to enhance learning outcomes and instructional efficiency. Through careful monitoring of this deployment, we quantitatively evaluated *GradeSuite*'s effectiveness in streamlining grading workflows, improving student success metrics, and supporting the implementation of mastery learning policies. Our evaluation demonstrates that *GradeSuite* successfully enhanced both student learning experiences and course administration efficiency, while providing the necessary infrastructure to implement mastery learning at scale. The platform's impact was particularly evident in its ability to support student self-directed learning and significantly reduce administrative overhead for course staff. This evaluation will provide valuable insights into the platform's capacity to transform existing pedagogical policies through technology-enabled mastery learning approaches. During our analysis, we also recognized places where the microapps could be improved, and will seek to address them in later years.

Chapter 2

Background

While there is a substantial body of research on learning management systems (LMSs) and mastery learning as separate domains, their specific intersection remains underexplored. Nevertheless, certain existing dashboards and prior research initiatives have emphasized key aspects of mastery learning, which served as important sources of inspiration for the development of *GradeSuite*.

2.1 Prior Work

While some existing dashboards such as Khan Academy's module progress view [45] provide students and instructors with more detailed breakdowns and visualizations of conceptual mastery, most of these platforms remain proprietary. Alternatively, platforms such as D2L Brightspace's competency-based education [13] dashboard or PowerSchool's *Assess-*

ment Reports [1] that seek to build mastery-based analytics enable adoption of some mastery-learning grading techniques, they remain inflexible to integrations with pre-configured course-management technologies and only extend bare-bones customization for course-specific grading configurations. These feature-rich platforms, though, provide a much-needed resource to instructors seeking to leverage mastery learning policies in their classes to enrich student learning. *GradeSuite* is built on the premise that such policies should be easily accessible, prioritizing integrations with existing LMSs to provide instructors with the flexibility to implement custom, more equitable grading practices [18] without the need to entirely migrate to new systems.

One key issue with existing platforms is the lack of grading information collected by the platform required to detail conceptual breakdowns assignments used in mastery-based policies. For example, the bespoke "Concept Map" software developed at the Institute for Human Machine Cognition (IHMC) lacks the functionality to integrate with mastery learning [9]. To combat this, platforms such as SCALA [46] provide much more fine-grained data to the instructor by taking a different approach: focus on learning *analytics* rather than learning *management*. SCALA scrapes data from various sources including certain existing LMSs to provide instructors and students with "competency assessments." While this system provides key metrics for many mastery-focused policies, it does not allow instructors to directly configure their own grading schema to assign grades due to its *beyond-LMS* design.

One journal article details how LMSs could *"reinforce the learning process through online classroom environments"* [6]. However, another case study of LMSs used in higher education

institutions found *"one of the main barriers to institutional adoption and usage of specific commercial platforms is that administrators are not permitted to constantly modify the system to better fit user's requirements,"* indicating that despite their vast ability to improve course management, the difficulty in customization presents a challenge to their adoption [49].

Existing platforms that focus on instructor-defined metrics, such as Ross Strader and Candice Thille's Open Learning Initiative (OLI) Instructor Dashboard [50] promote mastery-based grading policies through yet another, unique approach: learning objectives. The OLI dashboard allows instructors to define *learning objectives* [52] for their courses, and then use them to track student progress towards mastery. This approach allows instructors to more closely align their grading policies with their course goals, but unlike *GradeSuite*, its learning objectives are not directly integrated with traditional assignment grading, requiring instructors to reframe their grading structure and policies to effectively leverage the OLI dashboard.

Despite the lack of mastery-learning focused integration with existing LMSs [25, 8], some universities have validated the need for these systems by using their own, course-specific platforms. One such course studied a sample set of 29 students and found that their dashboard *"significantly enhanced students' self-control in terms of academic persistence compared to the control group"* [22]. Given the success of their implementation, we believed it was necessary to develop a platform that integrated seamlessly with existing LMS architectures while still providing the flexibility instructors needed to implement dynamic grading policies.

Additionally, empirical studies have established that mastery-based policies, such as

definitive deadlines with flexible extension policies, can significantly enhance student performance and the overall learning experience [33, 35]. Many of these policies specifically benefit *"weaker"* students [40] or underrepresented minorities [54] who may require additional resources to demonstrate adequate or equivalent course competency.

In a foundational paper on mastery learning, the author asserts that students *"[...] having time allowed for learning is the key to mastery"* [5]. Traditionally, this is addressed through extensions of assignment deadlines; however, one challenge to this approach is that a final grade needs to be calculated at the end of the term, meaning students are only able to work towards mastery for the duration of the course offering [23]. This terminal deadline may limit some students' ability to fully achieve mastery, with their only recourse being to retake the entire course. As such, some courses have began offering post-term mastery learning opportunities, using the "Incomplete" grade as a way to indicate that *"the students' mastery of the material is incomplete"* [53], while allowing them to finish their remaining coursework after the term ends [23].

Allowing post-term work adds additional complexities for course staff. Previous research has demonstrated how learning environments entirely devoid of deadlines are unsuitable for effective learning [34]. Resultingly, students who opt to continue their coursework under the "Incomplete" policy must closely coordinate with teaching assistants (TAs) to track remaining coursework, deadlines, and submission protocols. TAs, in turn, must manually update grade books, continually recalculate final grades, and submit updates to the Registrar's Office. Such support often relies on extensive email exchanges, additional office hours sup-

port, and complex management of semester-specific spreadsheets. This fragmented process introduces significant risk of miscommunication and grading errors, incurring unnecessary administrative overhead.

GradeSuite builds on prior development work done by aggregating and expanding existing projects that motivate use of mastery learning policies in the classroom. Namely, *GradeView* and *Instructor Dashboard* are based on works developed to address challenges with the User Interface (UI) of mastery learning dashboards [3]. Similarly, *GradeSync* is a development effort that builds upon existing work to synchronize course materials and avoid course staff the lengthy process attributed with handling incomplete students [4].

2.2 Motivation

When creating the *GradeView* dashboard, there were multiple existing LMSs and dashboards that we referenced both as demonstrations of exemplary applications for mastery learning as well as for areas that existing platforms fell short. In an informal survey of over ten introductory computer science instructors from institutions worldwide, we found that the added friction of implementing mastery-learning policies within existing LMS platforms was substantial enough to discourage instructors from exploring these policies altogether. Overwhelmingly, instructors indicated that the way their respective LMS handled mastery learning policies, including exam retakes, assignment extensions, concept visualizations, and overall grade calculation, was far from sufficient.

More specifically, our initial user studies for the *GradeSuite* project targeted instructors and course staff at UC Berkeley, who necessarily adopt Canvas [10] — one of the most widely used collegiate LMSs — into their instructional workflow due the university's native support at the institutional level. During our preliminary interviews, we quickly identified several core issues with the platform's support for mastery-based policies. Notably, Canvas entirely lacks support for *"clobbering"*, a grading policy where later demonstrations of conceptual mastery can be used to overwrite previously lower scores written to the grade-book only to indicate that the student's mastery was still in progress at the time [24]. Additionally, in order for courses to leverage customized grading practices, Canvas requires instructors to download course data and manually calculate custom grading metrics. Not only does this incur a significant amount of administrative overhead, but it also means that students are not able to directly track their progress in the class through traditional LMS features.

Yet another challenge for instructors at UC Berkeley using Canvas in mastery learning oriented classes is that provisioning extensions on a per-student basis is extremely cumbersome, requiring excessive amounts of time from course staff due to its complex, multi-step process. In order to provide a specific student with an extension, Canvas requires the creation of duplicate exam instances and custom deadlines for each individual student. While Canvas supports basic mastery learning functionality for tagging mastery levels [28], the aforementioned feature gaps introduce significant friction, effectively hindering instructors' ability to implement mastery learning in its entirety. In an interview with BJC course staff, they overwhelmingly mentioned that supporting mastery learning policies was a significant

challenge. Specifically, they stated that the process of updating grades was *"[...] painful, manual, and error-prone."*

Another key mastery learning policy we aimed to support was the use of "Incomplete" grades, as previously mentioned. In the past, handling an incomplete grade was a rare, one-off occurrence. According to interviews with CS10 course staff, these isolated cases were relatively easy to manage. However, implementing "Incomplete" grades as a formal policy significantly increased administrative workload. While the policy is intended to ensure equitable mastery learning opportunities for all students [23], its scalability remained a major challenge.

Our collaboration with CS10 course staff served as a foundational point for developing our system. In response to informal survey results, requests from CS10 staff, and motivation to spearhead the adoption of mastery learning, we aimed to design a platform with mastery learning centered at its core. Throughout the development of *GradeSuite*, we continually communicated with relevant staff members to ensure that it would provide all of the necessary features to enable a wide range of mastery learning policies while simultaneously being flexible enough to support future policies.

Chapter 3

Organizational Structure

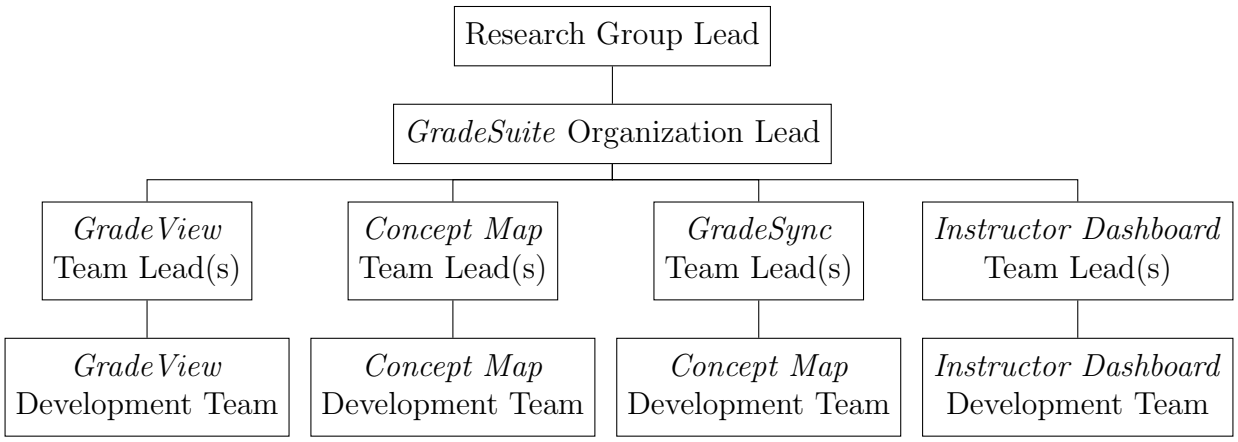


Figure 3.1: *GradeSuite* Organizational Structure.

The *GradeSuite* project has evolved significantly since its inception as an individually-developed dashboard three years ago. Initially conceived as a front-end dashboard for visualizing Google Sheets grade data without additional persistence requirements, the project has

since grown into a comprehensive suite of microapps. This expansion necessitated the implementation of a robust organizational structure to effectively manage concurrent development and deployment processes.

Following its initial unveiling and call for development support, the project attracted four dedicated engineers, establishing itself as one of the groups' largest development teams at the time. *GradeSuite* expansive impact horizon coupled with its robust design, multi-faceted development framework, and complex systems architecture further captivated researchers seeking to explore industry-grade technologies and best practices; just one year following *GradeSuite*'s proposal, the project expanded to six contributors focused on UI development and early microapp architecture. As the project continually gained traction, eliciting a variety of complex, mastery-focused feature requests, members of other teams flocked to the project, realizing the suite's potential for immediate impact. Currently, in order to support the project's growing scope, *GradeSuite* maintains a robust development team of over twenty engineers across its various microapps, reflecting its substantial growth and increasing complexity.

3.1 Chain of Command

To facilitate efficient development and deployment across microapps, we implemented a hierarchical organizational structure within *GradeSuite*, as illustrated in Figure 3.1. Notably, in order to best inform *GradeSuite*'s mastery learning, a teaching professor and pioneer

of mastery learning policies at UC Berkeley served as the research group lead while the *GradeSuite* organization lead — a founding engineer — managed and guided the project's technical development. While this structure defined clear reporting relationships, it was specifically designed to promote open communication across all organizational levels; both the research group lead and *GradeSuite* organization lead conducted weekly meetings with their entire reporting chain, fostering an inclusive environment where all team members could contribute meaningfully to discussions.

A key objective of *GradeSuite*'s organizational design was to establish an effective communication pipeline between technical and non-technical stakeholders. The organization lead functioned as a crucial intermediary, coordinating with individual microapp leads to track project milestones while ensuring alignment with broader organizational objectives. Moreover, given their technical expertise and comprehensive understanding of *GradeSuite*'s unified microapp architecture, the organization lead served as the primary liaison for cross-application concerns, facilitating seamless integrations and migrations between different components of the system.

GradeSuite's organizational framework enabled rapid assessment and resolution of development blockers through direct engagement with relevant teams, while maintaining comprehensive stakeholder involvement. The structure similarly streamlined the approval process, allowing decisions to be expedited directly to appropriate authorities without unnecessary hierarchical traversal.

3.2 Microapp Development Teams

Each of *GradeSuite*'s microapps was individually developed and maintained by its own research team. The size of each respective team depended primarily on the semester's planning session by assessing the magnitude of each team's deliverables. Teams with large epics (an agile framework term for "project with a large scope" [36]) typically had five to six developers; those with smaller scope (e.g., maintenance) only had two or three.

Each microapp's development team operated under the guidance of one or two team leads, with the number of leads scaling proportionally with team size. These leads worked in close coordination with the organization lead to structure projects into manageable epics and establish achievable timelines. As the primary representatives of their teams, leads participated actively in both organizational and research group meetings, ensuring effective communication of team progress and concerns. To maintain consistent project momentum, leads conducted regular stand-up meetings and check-ins throughout the week, facilitating real-time progress tracking and swift resolution of team-specific challenges.

Mid-Project Reorganization

In the project's early stages, team formation followed a self-selection model where developers chose their microapp assignments based on personal interest and subsequently managed their time allocation. While this approach proved effective during early development, its limitations became evident mid-semester as resource distribution became increasingly un-

balanced across teams. This observation led to the implementation of a comprehensive team reorganization initiative.

The reorganization strategy incorporated multiple factors beyond immediate project requirements, including individual developer expertise and familiarity with specific microapps. Senior team members and those with cross-application integration experience were identified as particularly versatile resources, capable of being reassigned across teams to optimize resource distribution relative to delivery commitments. Under the organization lead's direction, the transition period spanned approximately two weeks. Despite this initial investment in team restructuring, the enhanced operational efficiency achieved through improved resource allocation yielded significant productivity gains within the following month alone.

3.3 Management Technologies

In order to manage the entire organization, the organization lead configured a variety of different technologies utilized by developers and management alike. These technologies spanned from communication standards to project planning tools. Functionally, these tools allowed the entire *GradeSuite* organization to rapidly develop new features in line with higher-level organization incentives such as grant timelines or adoption candidates interested in deploying the suite (or individual microapps) for their own use.

GitHub

GitHub [7] is overwhelmingly the most popular file host developers use for storing and versioning their code, so we adopted the technology from the get-go. Since the project was initially just *GradeView*, we started with a monolithic structure, where all source code was stored in a single repository. While this made navigating and modifying early projects simple, as the project blossomed in size and microservices, we realized that continued expansion of this monolithic design would not be feasible in the long term.

Early Design and Expansion Challenges

Notably, we ran into issues with developers accidentally modifying code from other projects or breaking strict abstraction barriers, such as importing files from other microservices into the incorrect services or applications. Moreover, the monolithic design meant initial microservices did not have isolated environments to develop and test in, leading to an over-reliance on integration and validation testing.

To combat the many issues borne from the monolithic design, we decided to revise the development plan for an entire semester, prioritizing maintenance and structural refactoring efforts at the cost of feature development. During this maintenance semester, we decoupled the services into their current "microapp" structure by removing intersecting dependencies. Similarly, we focused on extracting out different microservices within each of the microapps, enabling us to separately develop each service while also ensuring that all of the service's

dependencies were localized to its own project directory.

GitHub Organization

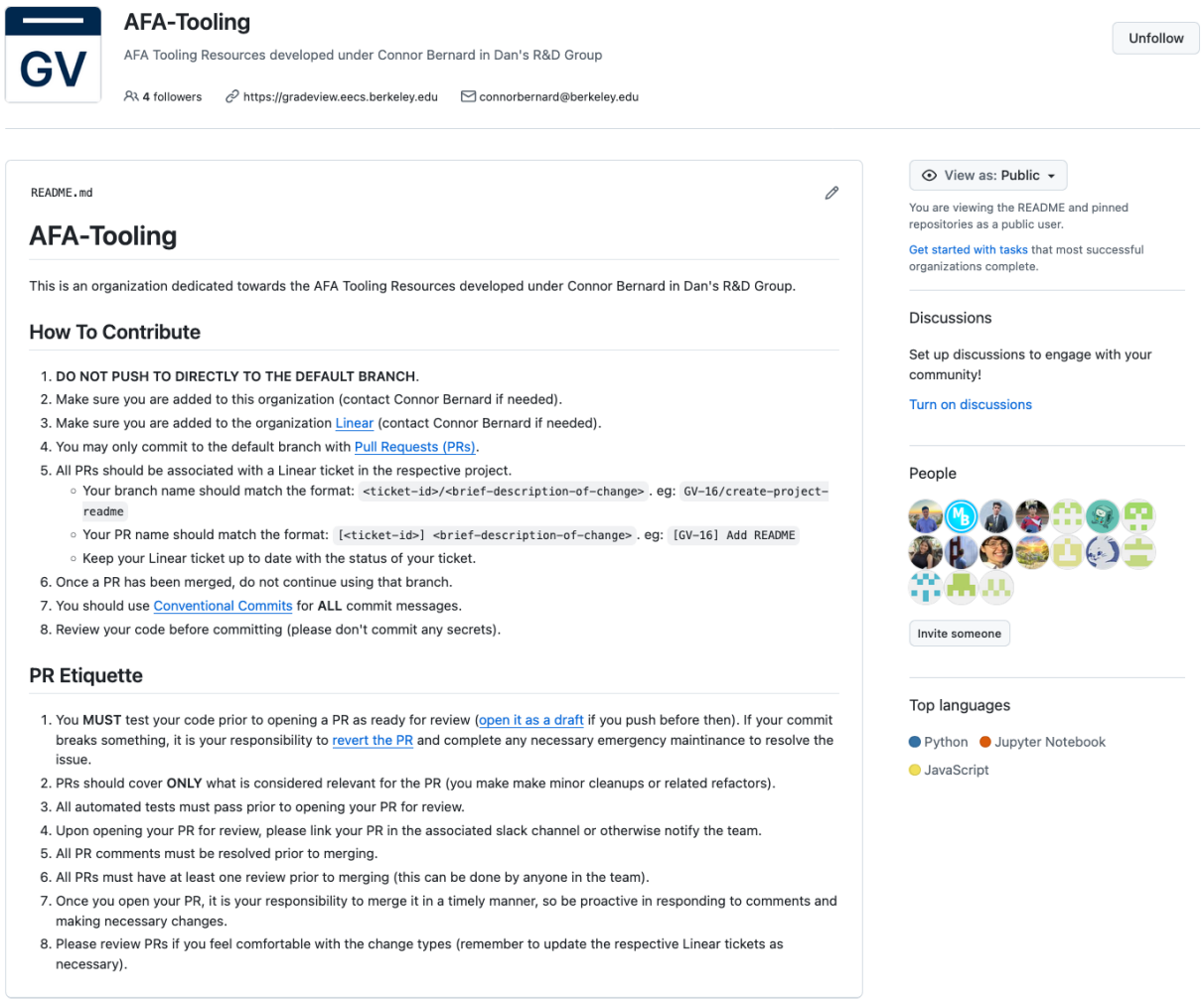


Figure 3.2: *GradeSuite* GitHub Organization Landing Page.

While breaking out the microapps into separate repositories, we realized that we needed a way to centralize our repositories in GitHub, since individually owned and managed repos-

itories was not realistic. To do this, we created a GitHub organization specifically for *GradeSuite* [2]. Not only did this allow us to localize our projects, but it also enabled key access control and management functions such as global branch protection. Moreover, the unified GitHub organization provided a centralized location for organization-wide documentation and shared scripts/utilities. Notably, the organization onboarding process and bylaws live in the GitHub organization's README.md, shown directly on the organization's GitHub landing page (Figure 3.2[1]). Not only did this serve as a reminder for best practices and contribution guidelines, but it also ensured that important debugging and disaster remediation plans were easily accessible in a known location.

Pull Request Etiquette for Documentation

One of the most important organizational decisions we made to ensure clean and maintainable code across *GradeSuite*'s several long-lived microapps is the rigid PR etiquette. These guidelines were strictly enforced; non-emergent changes in conflict with the guidelines were even automatically dismissed without review, forcing developers to update their PRs to comply with the guidelines. The strict guidelines in combination with a clear pull request (PR) template (Listing C.1) established baseline standards for implicit, persistently up-to-date code documentation, accessible through git's `blame` subroutine [26]. These standards assisted continuous, rapid development by mitigating the concern of outdated documentation that misrepresented the current state of the codebase.

[1]The original GitHub organization's name was "AFA Tooling," spanning from the broader research group's name "A's for All" [23].

Beyond documentation of code changes through PR bodies, the pull request template likewise required developers to link the ticket they were doing work under. Further, the ticket's identifier was included in the branch name and as the prefix of the PR name, for example a valid name might be `[GV-10] Migrate Concept Map to APIv2`. Not only did this enable key workspace automations in our ticketing platform that updated the status of tickets and linked the PR to the ticket, but it also ensured developers looking back at previous code changes were able to dive into more in-depth explanations of *why* a specific change happened, even being able to traverse up to the change ticket's respective epic or parent task.

Slack

Slack served as the primary communication platform for the entire organization, functioning as both the central hub for interpersonal communications and an automated notification system for project updates and deadlines. While synchronous team communications such as stand-up meetings were conducted in person or over video calls, we leveraged Slack's extensive integration capabilities with development platforms such as GitHub, Jira, and Linear to maintain automated oversight of ticket progression and pull request status. The platform also played a crucial role in our operational reliability strategy, serving as one of several redundant channels for on-call notifications to ensure prompt response and resolution of production incidents.

One of the most important workflows we configured for Slack was our integration with GitHub. As soon as a pull request was opened for review in one of the *GradeSuite* GitHub organization's repos, the respective slack channel was sent a message with a link and a short preview of the PR. This allowed developers on the team to easily preform code reviews as soon as the PR was ready for review. Likewise, when the PR was eventually merged in, the GitHub integration bot sent a respective notification not only to the team's channel, but to a larger organization-wide channel such that any potential related issues with downstream microapps could be diagnosed and resolved swiftly.

The integrations with our project management platforms (Jira and Linear) represented another critical component of our communication infrastructure. These integrations facilitated comprehensive project visibility across teams, enabling developers to maintain awareness of parallel development efforts and their respective roadmaps. Additionally, they enhanced accountability through automated ticket tracking and status updates, preventing task stagnation while ensuring efficient progression of work items through the development pipeline.

Ticket Tracker

Our project management system represented a cornerstone technology in *GradeSuite*'s development infrastructure. Given our adherence to agile methodologies, the selection of an appropriate ticket tracking platform remained crucial for enabling rapid development within

our agile framework [36].

Jira

During the project's initial phase, Jira [30] emerged as an attractive solution due to its comprehensive *freemium* licensing model, which provided extensive platform functionality at no cost for teams of up to ten members. However, upon exceeding this threshold, the platform's pricing structure proved prohibitively expensive, with our educational institution discount of 75% still resulting in monthly costs of over $100. While the platform's capabilities justified this expenditure in the short term, a cost-benefit analysis revealed that alternative solutions, such as Linear [32], offered comparable functionality with significantly more favorable pricing models for the growing organization.

Linear

Realizing that Jira was not a viable long-term solution for our team, we decided to migrate to our current project management platform, Linear. Linear provided much of the same services as Jira, but without the seat-number-based payment plan, allowing our organization to use it for our specific needs entirely free of charge. Unfortunately, since we already had our systems set up and configured for Jira, it meant spending roughly two weeks migrating the existing configuration, tickets, and integrations to Linear. On top of the migration costs, it also affected everyone in the organization as they spent valuable time familiarizing themselves with the new platform.

Despite the aggregate time-cost of migrating to Linear, we decided the that monthly cost of Jira sufficiently offset the one-time cost to migrate to Linear, so we moved forward with the project. Luckily, Linear provided a helpful migration strategy and tool to do a large amount of the work copying tickets from Jira to Linear. Unfortunately, it did mean forgoing the pre-existing automations and project boards that we had configured specifically within Jira. Another challenge we faced with the migration was that Linear's free plan only allowed for two distinct teams, whereas each microapp could have its own team in Jira. In order to mitigate this issue, we simply had different projects for each microapp in which the teams' respective tickets resided.

Luckily, like Jira, Linear provided light-weight integrations with both GitHub and Slack so that our existing pipelines could remain the same post-migration. This did cost some configuration time to set up prior to the organization's migration to Linear, but it made swapping to Linear a seamless process for existing developers in terms of their overall workflow.

3.4 Key Leadership Takeaways

Throughout the development and management of *GradeSuite*, several critical leadership insights emerged that proved instrumental to the project's success. First and foremost was the importance of maintaining forward momentum while remaining adaptable to change. This manifested in various ways, from our willingness to reorganize team structures when inef-

ficiencies became apparent to our strategic platform migrations when cost-benefit analyses warranted such transitions.

A second crucial insight was the value of establishing clear communication channels while maintaining flexibility in their usage. While our organizational structure provided a formal hierarchy, the deliberate decision to encourage cross-level communication during weekly meetings fostered an environment where innovative ideas could emerge from any level of the organization. This approach proved particularly valuable during technical integrations between microapps, where developers could directly address concerns with relevant stakeholders rather than navigating multiple organizational layers.

The importance of documentation and process standardization emerged as another vital lesson. Our strict PR guidelines and templating requirements, while initially seeming bureaucratic, proved invaluable as the project scaled from 4 to over 20 developers. These standards not only facilitated knowledge transfer but also significantly reduced the cognitive overhead for code reviews and maintenance tasks.

Perhaps most significantly, we learned the importance of balancing technical debt against feature development. The decision to dedicate an entire semester to maintenance and structural refactoring — moving from a monolithic to microservice architecture — initially appeared to slow progress. However, this investment in architectural integrity ultimately accelerated development velocity and improved system reliability. This experience reinforced the principle that technical leadership often requires making difficult trade-offs between short-term gains and long-term sustainability.

Finally, we discovered the critical nature of team composition and the need for strategic resource allocation. The mid-project reorganization demonstrated that, while self-selection of projects can boost initial motivation, optimal team performance requires thoughtful consideration of both technical requirements and individual capabilities. This balance between team-member autonomy and organizational needs became a cornerstone of our management philosophy.

Chapter 4

Application and System Design

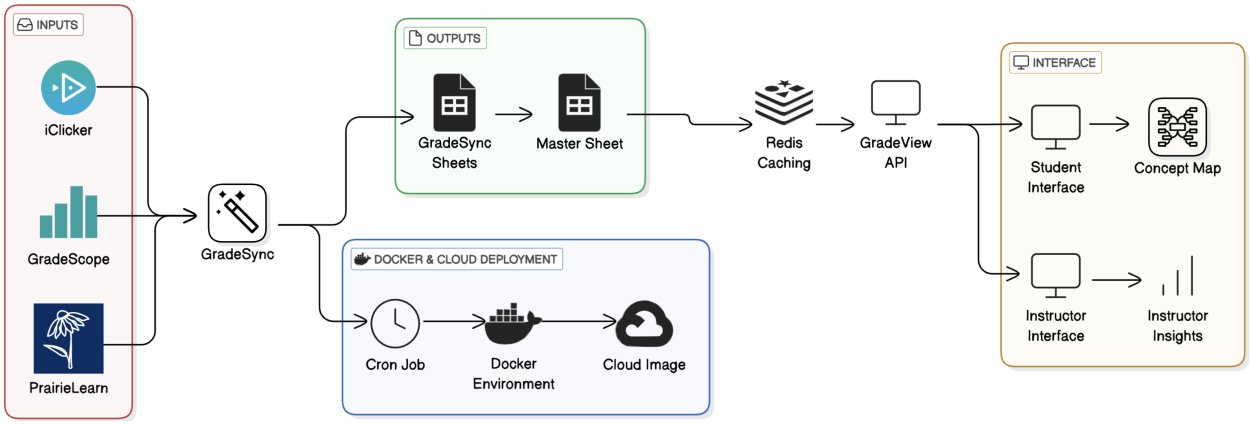


Figure 4.1: *GradeSuite* Data Flow.

GradeSuite is a microservice architecture hosted on Google Cloud developed and maintained by individual teams. It is currently comprised of four main microapps: *GradeView*, *Concept Map*, *GradeSync*, and *Instructor Dashboard*, each containing one or more microservices that are independently developed, maintained, and deployed.

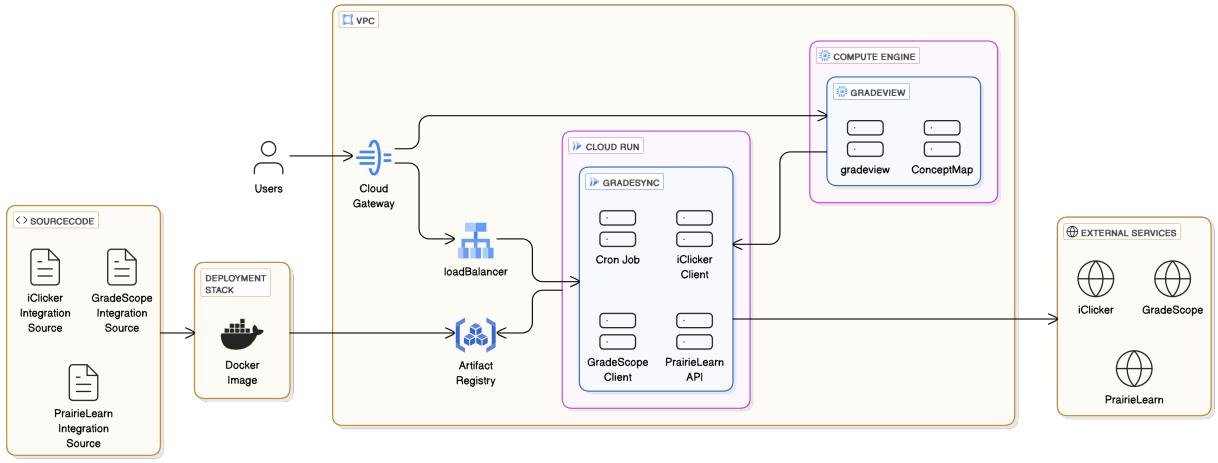


Figure 4.2: *GradeSuite* Architecture.

Each of the different microapps consists of one or more Docker containers configured to network with each other to provide all application functionality [17]. The apps interact with each other using their respective public/private Representational State Transfer (REST) Application Programming Interfaces (APIs) and/or through individually built integrations with other platforms. Figure 4.1 illustrates the end-to-end data flow -— from input to processing to output, while Figure 4.2 depicts the *GradeSuite*'s currently deployed cloud architecture.

4.1 *GradeView*

The *GradeView* microapp is the primary way that instructors and students alike directly

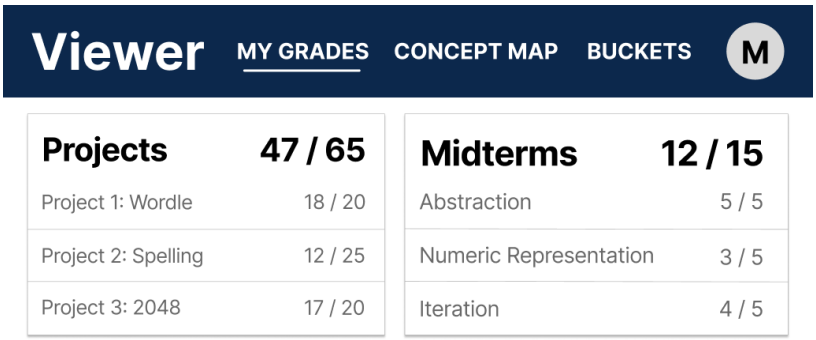


Figure 4.3: *GradeView* Student Interface.

interact with the suite. It is functionally comprised of four primary microservices: a reverse-proxy service that orchestrates and balances requests to each of its microservices, a front-end service that serves as the centralized interface of *GradeSuite*, a public API for all non-UI related requests, and a Redis caching layer to minimize transactional load.

One of the primary focuses of *GradeView* during its conception was to minimize (or altogether abstract away) the need for an external database. The motivation for this decision was to reduce data-storage overhead and promote platform integration over individual data management so instructors using the platform could leverage existing course infrastructure without having to migrate to new systems. To do this, *GradeView* relies on a Google Sheet as the sole "source of truth" for grade data. During our preliminary survey of instructors at UC Berkeley, we found that instructors with courses geared towards mastery-learning had complex grading schemes configured through the Google Sheets formula to facilitate related policies. As such, our goal was to architect *GradeView* to conform to instructors' existing Google Sheets instead of a rigid, bespoke database schema.

Reverse-Proxy Service

The proxy service for *GradeView* consists of a docker container with an NGINX [41] base image. It is configured to proxy public traffic to the other microservices in the application stack. It acts as the public gateway for all external connections and provides Transport Layer Encryption (TLS) for all HTTP requests including all internal API requests depicted in Figure 4.2. While this service was initially implemented for *GradeView*, it was later expanded to include all of *GradeSuite*'s microapps as they built direct integrations with *GradeView*.

Front-End UI

The front-end UI, as shown in Figure 4.3, is what both users and instructors interact with when they use *GradeView* from a web browser. Users log into the service through a public web-URL using OAuth [42] which is then validated not only for institutional access but also for course membership. For a student to successfully log in, they must be listed as a student in the instructor's Google Sheet as shown the second column of Table 4.1.

Additionally, the underlying Google Sheet is the primary source for all of the key UI components of the dashboard Categories and topics are defined with headings, which can be configured and updated on the fly simply by editing the Google Sheet. Likewise, new assignments and concepts can be added to *GradeView* simply by adding additional columns to the sheet. Moreover, rows can be easily hidden, simply by deleting the category-level

Legal Name	**Email**	**Abstraction**	**Iteration**	**Algorithms**
CATEGORY	**CATEGORY**	**Quest**	**Quest**	**Midterm**
MAX POINTS	**MAX POINTS**	**2**	**4**	**8**
Doe, John	johndoe@berkeley.edu	2	3	7
Smith, Alice	alicesmith@berkeley.edu	1	4	8
Williams, Eve	evewilliams@berkeley.edu	2	4	8
Wilson, Bob	bobwilson@berkeley.edu	1	2	3

Table 4.1: Master Google Sheet for *GradeView*.

column header. For example, if configured with the example sheet in Table 4.1, *GradeView* would display two categories/assignments, *Quest* and *Midterm*, with the topics *Abstraction* and *Iteration* under the *Quest* category, and *Algorithms* under the *Midterm* category.

After logging in, students can view not only the assignments they have completed and their aggregate grade on the assignment, as they would in a traditional LMS, but also the conceptual breakdown for each assignment and how they are performing on each related concept. This detailed breakdown of how they are performing at an individual, concept level provides a much deeper insight into course progression than traditional LMSs. Here, students can better focus their efforts by selectively studying the topics they have yet to master.

Through a direct integration with the *Instructor Dashboard* microapp (section 4.4), logged-in course staff likewise can view detailed breakdowns of conceptual mastery per-student. This provides course staff with a better understanding of the foundational concepts students are struggling with, allowing them to better assist individual students. *GradeView*'s

frontend UI also informs course staff of how the class as a whole is absorbing material at the conceptual level; staff could then use this information to motivate prioritized concepts in lectures, assignments, and other course materials.

The front-end application is built with *React* [47], using *Material UI* [38] as its front-end framework to ensure both fluid functionality and native accessibility. When developing the front-end, we heavily prioritized accessibility such as by supporting assistive technologies including screen readers, as our target audience is comprised of students from all backgrounds.

API

The *GradeView* API serves as the central nervous system of the application, providing a secure and efficient interface for all data management and external integrations. Built as a versioned REST API [37], it implements robust role-based access control at the resource level to ensure the protection of sensitive grade data. This security model is particularly crucial given the educational context, where student privacy and data protection are paramount.

To ensure optimal performance and scalability, the API is implemented in Node.js [21] and follows modern ECMAScript Module syntax, maintaining consistency with the React frontend's development standards. This architectural choice not only simplifies the development process but also enables seamless integration between the frontend and backend components. Moreover, our API is containerized and deployed as a Node.js Docker image, allowing for dynamic scaling through load balancing to handle varying request volumes.

A key architectural decision was the development of a custom Redis facade that abstracts and standardizes all caching operations. This facade layer provides a consistent interface for the API to interact with Redis, encapsulating complex caching logic and patterns while ensuring proper error handling and data consistency. The facade implementation includes standardized methods for common operations such as cache invalidation, data synchronization, and batch processing, making it easier to maintain and extend the caching layer as the application evolves.

The API's design emphasizes both security and performance, with each endpoint carefully crafted to handle specific use cases while maintaining strict access controls. When users authenticate through the frontend, the API serves as the gatekeeper, validating permissions and providing access to the appropriate grade information and service-specific integration data. This centralized approach to data management ensures consistency across the application while maintaining the flexibility needed to support various educational workflows.

Redis Caching Layer

To increase the maximum load and minimize transactional latency, we re-architected our application to use Redis as a caching layer.

The Redis layer sits in between Google Sheets and the API, since we noticed that requests to Google Sheets consistently took the greatest amount of time to resolve. After introducing the caching layer, not only were requests able to be handled much faster, but the maximum processable load grew substantially.

During our initial release, we realized that the high volume of requests to Google Sheets was a glaring bottleneck, even with a dedicated service account provisioned with an increased rate-limit. We noticed that connections to Google Sheets were being throttled due to rate limit violations during peak operational hours (post-exam after releasing grades). The Redis instance resolves this by abstracting Google Sheets from the API entirely. To do this, we use individual cron [14] jobs, programs configured for scheduled execution, to refresh and update the cache on preset intervals ensuring that the rate limit, set directly by Google Sheets, would never be exceeded.

When planning *GradeView*, we wanted to provide real-time data to the student as best as possible; as soon as the Google Sheet was updated, so would the *GradeView* UI. As such, the cron jobs are optimized to run efficiently and often, ensuring minimal refresh latency. Given this optimization, we also configured the Redis [48] layer (and related cron jobs) as Docker images, allowing for them to be easily spun up and executed on the fly. This architecture is consistent with our plan to avoid an application-specific database since it functions as a cache rather than as a stand-alone database. It also means that the Redis instance can likewise be balanced as any other application.

4.2 *Concept Map*

The *Concept Map* microapp is a key component of *GradeSuite* that provides students and instructors with a clear visual representation of the course's conceptual structure and each

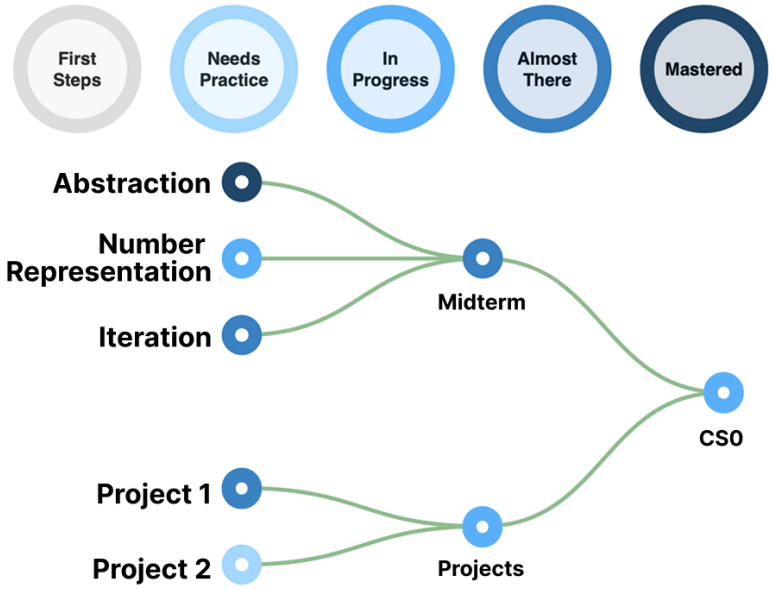


Figure 4.4: *Concept Map* Diagram.

student's mastery progress, as illustrated in Figure 4.4. Traditional LMSs typically display grades at the assignment level or as an overall aggregate, which can make it difficult for students to track their understanding of specific topics. This lack of granular insight often prevents learners from identifying the concepts they have mastered and those requiring further study. Additionally, many concepts in a course serve as foundational prerequisites for more advanced material, making it essential for students to understand the logical sequence of topics and how assignments contribute to their conceptual development. The *Concept Map* microapp uniquely provides a visually interactive solution to these exact issues.

The *Concept Map* module focuses on individual concepts and their interrelationships, allowing students to quickly see where they stand on each topic and instructors to pinpoint

areas of confusion or misunderstanding. By visualizing how each concept connects to subsequent topics, the *Concept Map* microapp allows learners to plan their study schedules more effectively -— focusing first on foundational material and then moving on to more complex concepts once they have established a solid base. This approach fundamentally changes how students engage with course material, transforming the learning experience from a series of disconnected assessments into a coherent journey of conceptual mastery.

Early versions of *Concept Map* also included a feature that allowed instructors to define expected course progression. This enabled edges in the map to be colored based on whether or not concepts were taught, allowing students to pace themselves accordingly (see Figure 4.5). In order to make the *Concept Map* visualizations more dynamic, we later removed this feature, and instead opted to render only nodes and edges that had been taught, so as to not overwhelm students with information that was not directly relevant to their current learning journey.

Web Application

The *Concept Map* microapp is implemented as a Python Flask-based [20] microservice within the larger codebase of *GradeSuite*. Its architecture follows a three-tier design pattern that separates concerns between data management, processing logic, and presentation. The data layer is responsible for storing and retrieving concept definitions and mastery scores, while the processing layer manages the transformation of raw data into visualizable structures.

The presentation layer handles the rendering of the interactive visualization through a web interface, ensuring a clear separation of functionality that promotes maintainability and scalability.

The web interface of *Concept Map* is built using React [47] and D3.js [15], providing a rich, interactive experience for users. The interface supports sophisticated navigation features, including smooth zooming and panning capabilities that allow users to explore the concept map at different scales. Users can expand or collapse nodes to reveal or hide detailed information, creating a customizable view that adapts to their current focus and needs. Moreover, the visualization employs a force-directed layout algorithm that automatically positions nodes in a way that minimizes edge crossings and optimizes readability [12]. This algorithm takes into account both the hierarchical structure of the concepts and the strength of their relationships, creating a layout that naturally reflects the conceptual organization of the course material. The resulting visualization is both aesthetically pleasing and functionally effective, allowing users to quickly grasp complex relationships between concepts and their current mastery status.

To ensure maximum accessibility and deployment flexibility, *Concept Map* can be deployed either as an integration through its API as is done with *GradeSuite* or as a fully interactive web application. The standalone deployment allows instructors to generate static HTML/JS/CSS files that can be hosted on any web server, making it particularly useful for environments with limited infrastructure. Additionally, it supports static generation of concept map JPG images or PDF files as shown in Figure 4.5, leveraging Graphviz [19] as

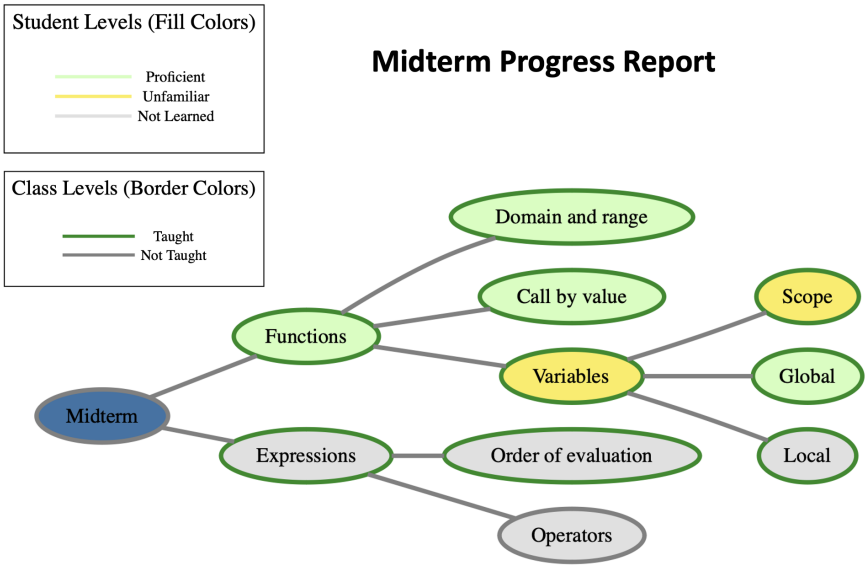


Figure 4.5: Static *Concept Map* Diagram.

its underlying visualization library. This dual deployment approach ensures that the *Concept Map* microapp can be integrated into a wide range of educational environments while maintaining its core functionality and visual appeal.

Syntax and Configuration

The configuration system for *Concept Map* is designed to be both powerful and accessible to instructors. Using a YAML-based syntax such as the one shown in Listing C.3, instructors can define complex hierarchical relationships between concepts with unlimited nesting depth. This configuration language supports not only basic concept definitions but also sophisticated relationships between concepts, including prerequisite chains and assignment

mappings. Moreover, *Concept Map* accepts JSON configurations as well, allowing instructors not familiar with our proprietary syntax to effortlessly adopt *Concept Map* into their existing workflow. Additionally, each concept can be enriched with custom metadata, such as learning objectives and associated resources, providing a rich context for both students and instructors. The system's flexibility allows for the representation of both simple linear progressions and complex, interconnected knowledge structures.

The configuration system's most notable feature is its ability to represent assignments as nodes alongside concepts, creating a direct visual connection between assessment and learning objectives. This integration enables learners to see exactly which assignments reinforce or assess particular topics, providing immediate context for their performance. The visualization uses an instructor-customizable color-coding system that to denote progression from areas of struggle to complete mastery, providing an intuitive visual representation of student progress. This color-coding system is dynamically updated as students complete assessments, creating a real-time feedback loop that guides their learning journey.

Data Processing Pipeline

The data processing pipeline in *Concept Map* is designed to transform raw configuration and performance data into meaningful visualizations. The process begins with configuration parsing, where the YAML-based concept definitions are transformed into an internal representation that captures both the hierarchical structure and the relationships between

concepts. This internal representation serves as the foundation for the subsequent stages of processing, which include mastery calculation and graph generation.

The *Concept Map* microapp generates an undirected graph structure that represents the complex web of concept relationships, with edges indicating both hierarchical and prerequisite connections. To calculate student mastery *Concept Map* aggregates student performance data across subtopics and assignments, taking into account both direct assessment results and inferred understanding based on prerequisite relationships. Additionally, *Concept Map*'s visualizations account not only for total score of child assignments, but also for each child's mastery level in order to best depict student proficiency and progress.

GradeSuite Integration

Concept Map is deeply integrated with the other components of *GradeSuite*, creating a cohesive ecosystem that supports both student learning and instructor insight. Its integrations with *GradeView* and *GradeSync* ensure that mastery scores are updated in real-time as new grade data becomes available, providing students with immediate feedback on their progress. This real-time updating is crucial for maintaining the relevance and accuracy of *Concept Map* as a learning tool. These integrations ensure that the visualization remains current with the latest assessment results while maintaining the conceptual organization defined by the instructor. The system's ability to connect assessment data with conceptual understanding creates a powerful feedback loop that supports mastery-oriented learning. This comprehen-

sive integration ensures that *Concept Map* remains a dynamic and relevant tool throughout the learning process, providing both students and instructors with valuable insights into the learning journey.

Concept Map fosters a mastery-oriented mindset by making conceptual gaps visible at a glance. Its visual nature and interactive features make it an effective tool for both students seeking to understand their learning progress and instructors looking to identify areas where students need additional support. The system's architecture and deep integration with other components of *GradeSuite* create a powerful platform for supporting mastery-based learning in complex educational environments.

4.3 *GradeSync*

GradeSync serves as a microservice-based microapp for data synchronization between Gradescope [27], PrairieLearn [44], and iClicker [29], leveraging Google Sheets as an intermediary for processing and visualization. The architecture follows a modular, cloud-based design to ensure scalability, reliability, and ease of deployment. It is neatly integrated with *GradeView* to allow instructors the ability to easily manage its configuration as shown in Figure 4.6.

Data Pipeline

The system is structured as a pipeline to handle grade synchronization and student tracking efficiently. It begins by extracting student grade data from three major LMS platforms:

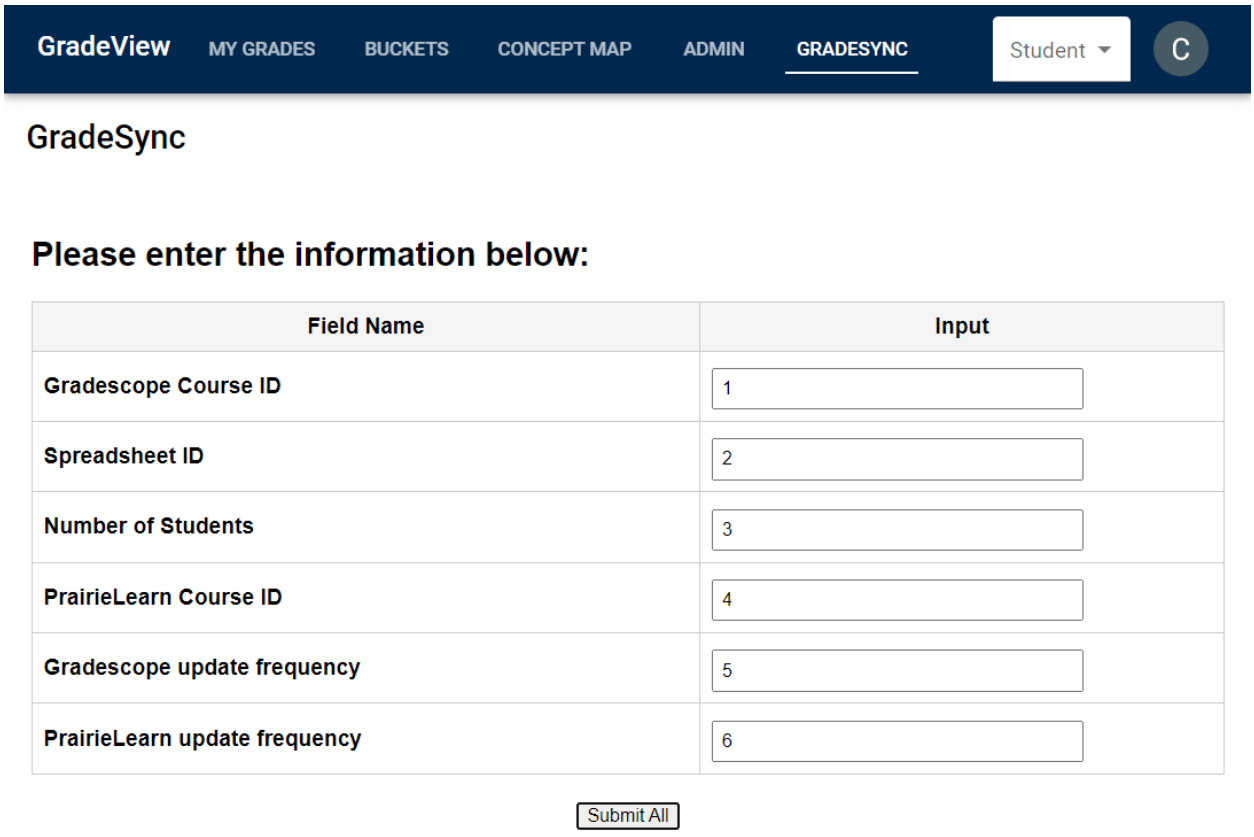


Figure 4.6: *GradeSync* User Interface.

Gradescope, PrairieLearn, and iClicker. Each of these LMSs expose APIs or user access strategies that can be automated to allow *GradeSync* to fetch, update, and process grades programmatically. By configuring the microapp at the beginning of the semester with respective service account credentials or API keys, *GradeSync* can automatically retrieve and process grade data from each platform. The extracted data is then processed into Pandas dataframes through custom transformations and subsequently pushed to pre-configured or automatically generated Google Sheets for ongoing tracking and analysis. When integrated

	Gradescope		**PrairieLearn**	
Name	**Proj1**	**Proj2**	**Midterm**	**Final**
Alice	19	14	64	76
Bob	12	16	43	85
Catherine	20	17	78	110
Mary	18	12	70	118

Table 4.2: Student Scores Aggregated across Multiple Platforms.

with the *master* Google Sheet used by *GradeView*, the master sheet can then reference the processed grade data by importing it with the `IMPORTRANGE` command, allowing the instructor to configure any number of sheets formulas to calculate and track student mastery scores.

The core processing logic of *GradeSync* is encapsulated in a set of scripts responsible for data retrieval, transformation, and synchronization. Table 4.2 illustrates a simple example of the aggregation of scores across multiple platforms into a centralized dataframe. These scripts are scheduled to run periodically using cron jobs to ensure automatic updates (e.g., hourly synchronization). The system is containerized with Docker, enabling portability and seamless deployment across different environments. Google Cloud serves as the hosting platform, ensuring high availability, scalability, and fault tolerance.

Data Storage

Google Sheets functions as an intermediate data storage solution in the *GradeSync* architecture, serving as a centralized repository for synchronized grade data from multiple learning

management systems. This choice aligns with our broader architectural goals of maintaining flexibility while ensuring data security and accessibility.

The decision to use Google Sheets as our primary storage solution was driven by several key factors. First, it provides a familiar interface that allows instructors to leverage existing spreadsheet expertise for custom grade calculations and analysis. Second, its robust API and real-time collaboration features facilitate seamless integration with our microservices architecture. Most importantly, in compliance with UC Berkeley's data storage policies [16], we ensure that all student grade data is stored exclusively in Berkeley-owned or licensed Google Sheets and servers, maintaining strict institutional data governance standards while providing the necessary functionality for our mastery-based learning platform.

Configurability

The system's high level of configurability enables professors and teaching assistants to seamlessly initialize and customize integrations by specifying input parameters such as a GradeScope course ID, PrairieLearn course ID, iClicker course ID, and a Spreadsheet output ID. With a single click, a Docker container configured for a specific course and semester is deployed to the cloud. Additionally, multiple deployments can be effortlessly initiated by defining new configurations, enabling these cloud services to run in parallel. This ensures that incomplete student data can be automatically processed beyond the original semester, as data from multiple semesters is actively pulled and processed. Instructors can customize grading formulas in Google Sheets to continuously calculate students' overall grades as well,

taking advantage of *GradeSync*'s rapid refresh rate to reduce the amount of time required to calculate grades.

Ultimately, *GradeSync* provides the underlying infrastructure to support mastery learning based grading policies through the integration of data from multiple platforms and customization grading formulas that accurately reflect student mastery.

4.4 *Instructor Dashboard*

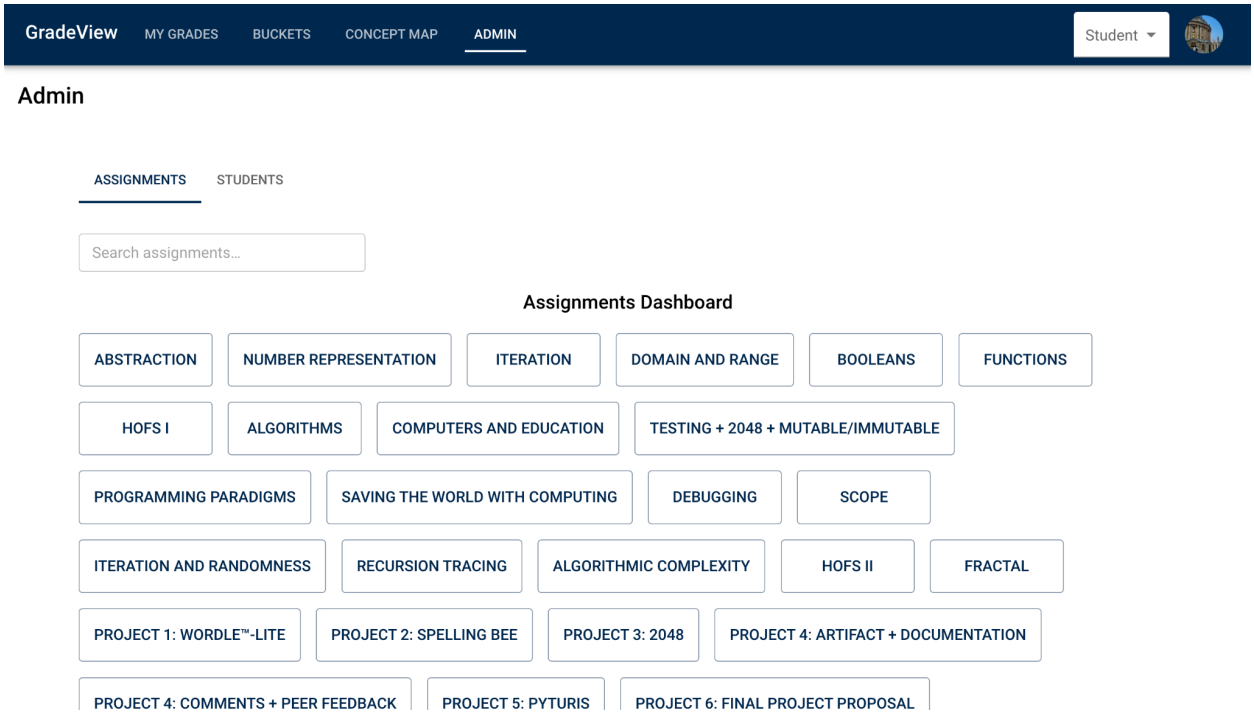


Figure 4.7: *Instructor Dashboard* Concepts Overview Page.

Although the first version of *GradeSuite* (consisting only of *GradeView* and *Concept Map*) was initially developed with students in mind, we quickly realized that instructors would also benefit from a dashboard to view course-wide mastery data. As such, *Instructor Dashboard* was developed to provide instructors with a mastery-based perspective on course-wide student learning while integrating seamlessly with *GradeView*.

Core Metrics and Computation

The *Instructor Dashboard* microapp implements a multi-stage data processing pipeline built on Python, leveraging direct integrations with both *GradeView*'s API and Google Cloud Platform services. The system's primary function is twofold: first, it aggregates and processes student mastery data from *GradeView* to generate actionable insights; second, it analyzes platform-wide usage patterns through Google Cloud Logging and BigQuery to provide comprehensive analytics on user engagement and system utilization.

The data processing architecture employs the Pandas framework [43] to perform real-time analysis of incoming data streams. This choice enables efficient manipulation of large datasets while maintaining the flexibility needed for complex statistical computations. The system processes both structured mastery data from *GradeView* and semi-structured log data from Google Cloud, transforming them into normalized datasets suitable for analysis and visualization.

The visualization layer of *Instructor Dashboard* supports multiple rendering pathways to

accommodate different use cases. When integrated with *GradeView*, the system utilizes Material UI [38] to deliver interactive, web-based visualizations that maintain consistency with the broader *GradeSuite* interface. For standalone deployments or automated reporting scenarios, *Instructor Dashboard* can generate static visualizations using matplotlib [39], which are particularly useful for offline analysis or integration with external reporting systems. Additionally, all visualization capabilities are exposed through *Instructor Dashboard*'s REST API, enabling programmatic access to analytics and facilitating integration with third-party platforms or custom dashboarding solutions.

User Interface

Instructor Dashboard's user interface is accessible directly through *GradeView*, and it contains several primary views. The first is the concepts overview page, shown in Figure 4.7, which displays a report of all of the course's key concepts. Each of these concepts can be clicked to view a detailed breakdown of the concept's course-wide mastery, as shown in Figure 4.8. Depending on the individual instructor's configuration in Google Sheets, the "concepts" may also include assignments that are similarly displayed in *GradeView* and *Concept Map* respectively.

Another important feature of *Instructor Dashboard*'s UI is its ability to neatly display the mastery breakdown of each student in a tabular format (see Figure 4.9). This view allows instructors to quickly identify individual students who are struggling with certain

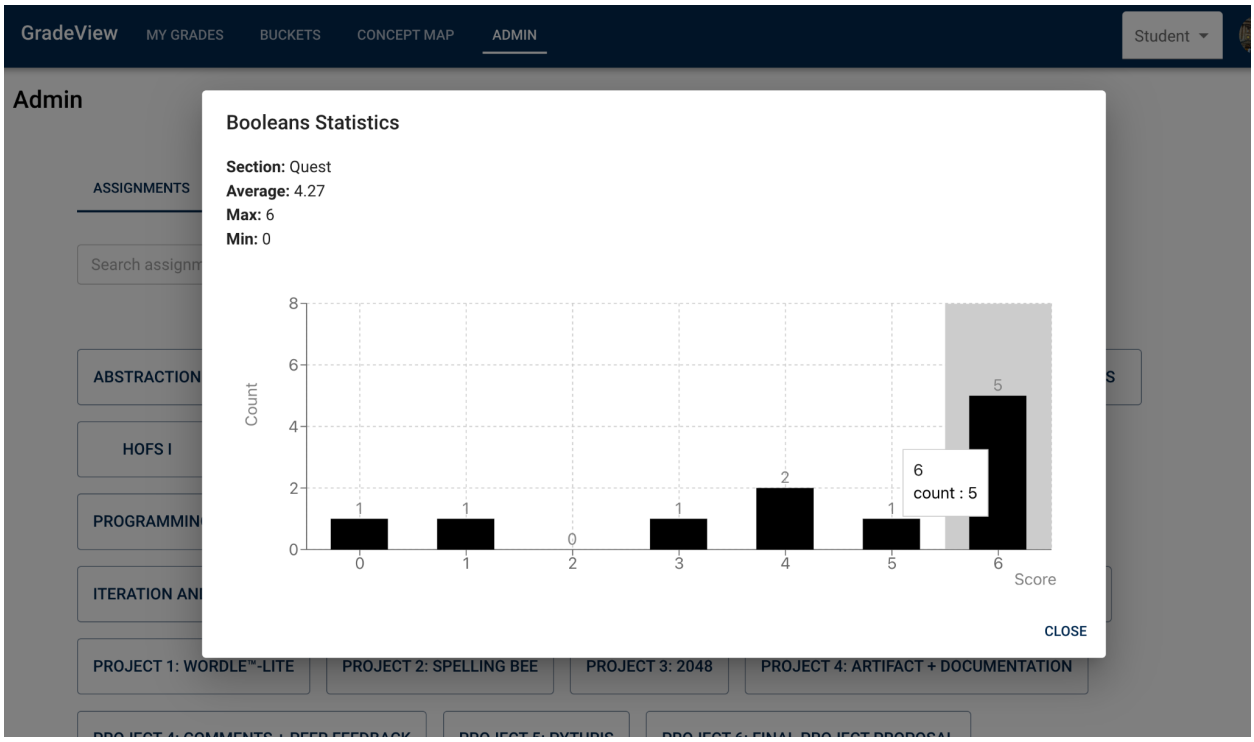


Figure 4.8: Concept Breakdown Page in *Instructor Dashboard*.

concepts, and provides them with a starting point for further investigation. Each column in the table is individually sortable which is likewise helpful in determining which concepts are most challenging for the class as a whole, and can be used for instructor support, such as by guiding the development of course materials.

Identity and Access Management (IAM)

A critical requirement for integrating *Instructor Dashboard* with *GradeView* was implementing robust access control to ensure that only authorized instructors and TAs could view and access the application. While *GradeView*'s existing access control infrastructure provided a

Student	Abstraction ↓	Number Representation	Iteration	Domain and Range	Booleans	Functions	HOFs I	Algorithms
There, Midway midwaythere@berkeley.edu	2	4	2	4	6	3	2	0
Smith, John johnsmith321@berkeley.edu	2	3	4	5	3	2	2	–
abc, def abcd@berkeley.edu	2	4	2	6	6	4	12	2
Perfect, Paula test@berkeley.edu	2	4	6	6	6	4	12	2
Mary, James test2@berkeley.edu	1	0	1	0	1	0	1	–
Started, Just juststarted@berkeley.edu	0	0	0	0	0	0	0	0

Figure 4.9: *Instructor Dashboard* Student Report.

foundation, two significant challenges needed to be addressed: role-based access differentiation and UI adaptation for instructor-specific functionality.

Initially, all access control was managed through the Google Sheet itself, which served as the sole source of truth for user roles and permissions. However, this approach lacked the granularity needed to distinguish between instructors and TAs, as well as the flexibility required for managing varying levels of access across different course components. This limitation became particularly apparent when implementing *Instructor Dashboard*, which required more sophisticated role-based access control.

Back-End Access Control

To address these challenges, we implemented a multi-layered authentication and authorization system. The solution leverages OAuth [42] for institutional authentication while introducing a new role-based permission model that expands upon the existing authorization

systems configured in *GradeView*'s API. This approach maintains backward compatibility while enabling fine-grained access control necessary for instructor-specific features.

In order to support the new, role-based access control, we had to modify the *GradeView* API config file to include role-based access as shown in lines 31–34 of Listing C.2. We deliberately decided to declaratively provision administrator access through the API config file to track access updates with our version control system, git. While rewriting our access control logic, we also had to revisit our existing authorization systems to ensure that requests by instructors (not listed in the Google Sheet) would be properly handled, since our previous instance would deny any requests by users not listed in the Google Sheet. Although this was a significant challenge, it was a great opportunity to refactor our existing authorization systems to be more robust and flexible.

To best integrate our new access control system into our API, we simply had to write a middleware function that would wrap all admin API requests. This functionality was relatively simple to implement, as all admin-specific API requests were routed through the `/admin` router, so the admin router itself natively integrated the middleware as a preliminary filter before further processing the request.

Front-End Visibility

The front-end implementation required significant architectural changes to support role-specific UI components and features. We introduced a new permission-based rendering system that dynamically adjusts the interface based on the user's role as provided by the

GradeView API, ensuring that instructor-specific functionality is only visible to authorized users.

This enhancement not only improved security but also streamlined the user experience by presenting each role with a tailored interface containing only relevant features and data. The modular design of this system allows for easy addition of new admin-specific features, supporting the growing needs of course staff while maintaining robust access control.

The way we controlled front-end role-based access control was with an extremely complex React user context that was provided at the top level of the *GradeView* application. The context contained a reducer function callback that would be called when the user logged in, and update the context with the user's information, including their role as specified by the *GradeView* API login endpoint. This not only allowed us to gracefully render student versus instructor interfaces, but it also meant that unauthorized API requests could be denied before they even reached the API, reducing load on the API itself. It is important to note, though, that this client-side check was used solely for performance benefits, as the API was already protected by the role-based access control implemented in the back-end.

Chapter 5

Deployment

GradeSuite is an extremely portable system, ensuring easy deployment across a variety of platforms. Frictionless deployment was an important consideration in our initial architecture, since we wanted to make sure that instructors could adopt *GradeSuite* into their workflow through much easier means than adapting their existing LMS to support mastery learning. Over the lifespan of the *GradeSuite* project, the deployment architecture changed several times, leading to its extremely dynamic design suitable for a multitude of deployment strategies. Namely, *GradeSuite* supports both bare-metal and cloud deployment with Google Cloud. Given it's bare-metal support, *GradeSuite* can easily be adopted to any cloud service provider by simply configuring the entire stack to run from within a virtual machine.

5.1 Containerization

From the outset, we decided it would be important to containerize the platform for development consistency and reliable deployments. To do this, we used Docker [17], overwhelmingly the most popular containerization platform. Each microservice was declaratively configured with its own image specification whether entirely through the docker-compose file, or through its own individual Dockerfile.

The compose file allowed us to localize the entire microapps' container stack for easy local deployment. This ensured that developer and deployment infrastructure was the exact same, such that there were no inconsistencies between production and development environments. Likewise, it allowed us to easily push built images up to our production cloud environment for easy execution and load balancing.

One initial challenge we faced with our entirely virtualized approach was the lack of data persistence for our microservices. Initially, we wanted to rely only on the upstream Google Sheet for all data access, thereby eliminating the need for a centralized persistence layer. Unfortunately, this this meant that all grade-data accesses made respective requests to the Google Sheets API which was not only a significant source of latency, but also proved to be a key bottleneck due to the API's rate limits. To subvert this issue, we decided to add a caching layer as a service — the Redis service — which would periodically make requests for the updated Google Sheet data and store it in the cache for quick access by the API microservice. Not only did this solve our latency and reliability issues, but it also allowed

us to more easily scale our platform horizontally, since the service itself can be spun-up on demand by pre-fetching the data before serving requests.

5.2 Early Deployment: Cloud to Bare-Metal Migration

The initial version of *GradeSuite* (Figure 5.1) was deployed directly to Google Cloud using Artifact Registry as a private image repository. From there, we configured Google Cloud Compute jobs to pick up the most recently deployed image (tagged with "`:latest`") and spin up compute instances for each microservice. In our deployment pipeline, we also declaratively configured orchestration of the compute instances to ensure consistent inter-service networking configuration as the platform scaled. Likewise, we set up our Virtual Private Cloud (VPC) with a dedicated gateway to allow for application-level load balancing and routing, while still providing the necessary isolation for each microservice.

After configuring the platform for Google Cloud deployment, our requirements changed, and we needed to move our entire platform to bare-metal. Doing so was relatively simple given the existing containerization configuration, so we were able to simply configure Docker on the deployment server and spin up the stack as we would in dev, but by setting the environment to prod. The biggest challenge with the bare-metal deployment infrastructure was in routing packets to the correct applications. Although *GradeSuite* has its own virtualized

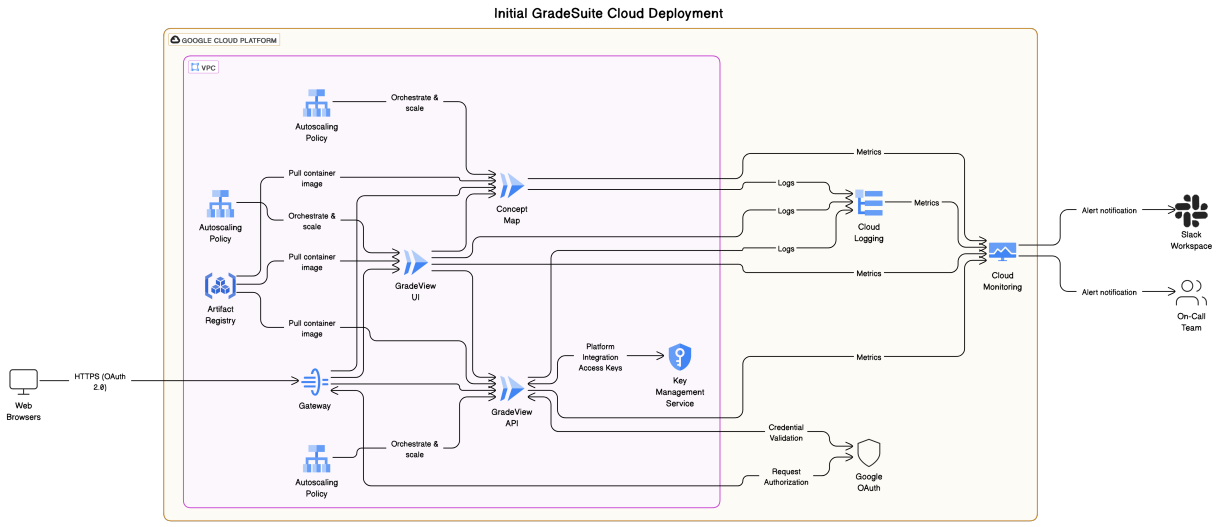


Figure 5.1: Initial Cloud Architecture of *GradeSuite* in Google Cloud.

reverse-proxy for all of its microapps, we found that the proxy service itself was not capable of directly integrating with the bare-metal host's web service leading to issues when the proxy service was opened on web ports 443 (HTTPS) and 80 (HTTP). To address this issue, we to established a secondary reverse-proxy on the deployment host. This also allowed us to configure our SSL/TLS certificates on the deployment machine once instead of automating their configuration at the *GradeSuite* reverse-proxy and in each additional microapp.

During our initial deployment of the development environment, we used one of our team member's personal servers as a bare-metal hosting solution. They locally forwarded ports 80 (HTTP) and 443 (HTTPS) to the deployment host on which *GradeSuite* listened for requests. While this strategy saved the overall team monthly costs for cloud infrastructure, it was not a viable long-term solution, and could not be used as a production host due to

data protection policies at UC Berkeley [16]. For this reason, we realized we would have to migrate back to the cloud before we could serve live grade data.

5.3 Updated Cloud Deployment

After realizing we would have to port all of our bare-metal infrastructure back to Google Cloud, we originally took it as an opportunity to clean up and improve our previous Google Cloud deployment strategy. Specifically, we knew that we wanted to implement better declarative infrastructure and access control, so we explored HashiCorp's Terraform [51], a commonly used Infrastructure-as-Code (IAC) language that would allow us to manage and version our cloud infrastructure from files in our git repository. Previously, we had used Google Cloud logic in our CI/CD GitHub workflow to procedurally configure and deploy our infrastructure.

As we further explored using Terraform for our IAC framework, we realized that there was far more infrastructure to deploy than what we had previously configured for the initial cloud deployment. Namely, in order to successfully configure our bare-metal solution, we had to stand up a proxy-service in a container capable of enabling Transport-Layer Security (TLS) for end-to-end secure communication between both internal microapps and external hosts using *GradeSuite*. This proxy service not only meant setting up a new microapp in our CICD pipeline, but it also meant entirely reconfiguring our networking architecture to route through the proxy as a gateway. While this solution was absolutely necessary for

the bare-metal deployment framework, porting it back over to the cloud caused issues with application-level load balancing and internal routing. Given the complex nature of our bare-metal solution, structuring it with Terraform was not as viable as we had initially planned, so we decided to approach it later on down the road. To subvert these issues, we attempted to deploy each of our non-private microservices in our Virtual Private Cloud (VPC) as publicly accessible applications, while persisting our virtualized proxy service which was tightly coupled to the *GradeView* and *Concept Map* microapps.

Our initial hope to migrate the proxy-as-a-service solution unfortunately did not go entirely as expected; although the proxy itself operated as anticipated, it did not provide the necessary certificates for proper TLS configuration on connections forwarded from our gateway. Moreover, this new approach required us to revise our Cross-Origin Resource Sharing (CORS) policies to support each of the new public endpoints for the individual microapps. Given these challenges, we decided to re-evaluate our migration plan entirely in hopes of encapsulating the entire deployment stack without the need to configure complex networking infrastructure.

Our final choice of infrastructure was quite trivial: use a virtual machine to spin up the entire stack using Docker as we had done on bare-metal. This not only preserved the entire configuration, but ensured that development environments would likewise be exactly the same as the deployment environment at the highest level. Unfortunately, this virtualized stack came at the cost of easily-configured, application-level horizontal scaling offered by Google Cloud's application load balancers. When evaluating *GradeSuite*'s potential peak

usage, though, we realized that the computational overhead of separately hosting and scaling individual applications was relatively similar to that of vertically scaling the virtual machine instance on automated load metrics, so we found that the vertically scaled approach was a valid trade-off for the development hours saved.

Learning from our migration of our early microapps back to Google Cloud, we ensured that all future microapps such as *GradeSync* were individually deployed and managed through their own respective cloud compute instances such as Google Artifact Registry and Google Cloud Run respectively. Specifically, for *GradeSync*, we use Google Cloud Scheduler to configure automated triggers to update core databases used by *GradeSync* in the background that function separate from the existing infrastructure deployed on the virtual machine.

5.4 Continuous Integration/Continuous Deployment (CICD)

A key goal in the development of our microapp infrastructure was to ensure that all applications and the stack itself were configured with proper CICD practices. Namely, we wanted to ensure that deployments could be done easily, and that testing could be run automatically. In order to do this, we used GitHub actions to run our test suites upon opening a PR, and required tests to pass prior to merging. Likewise, we used automated linters, smell-checkers,

and security audits, all of which were similarly required to pass before a ticket would even be reviewed by a team member. These application-level, non-framework-specific integrations were configured at the organization level in the overall GitHub organization to enforce compliance across all microapps.

Pull Request Review

Perhaps the most critical policy we enacted to ensure correctness of CICD strategy was the review process. While we relied on automated testing and code-coverage in later microapps, earlier implementations and apps lacked necessary testing, so those projects had a strict Pull Request (PR) review process: team members/code-owners were responsible for reviewing PRs to their own repos/projects, and were similarly responsible for maintaining any code that was published to their projects. This meant that co-authored commits or commits from other teams needed to be carefully parsed and understood by the adopting team prior to merging them in. Likewise, it ensured that all code-owners had a well-rounded understanding of how their entire application worked, adopting responsibility of different parts of their application through the review process.

Before merging any pull requests, all review comments needed to be addressed and resolved, and a terminal approval review was necessary. Likewise, if any of the code in the PR had been impacted by a more recent merge to master regardless of whether it resulted in a conflict, the PR would need to be updated and re-reviewed prior to merging. This strict

PR review process balanced ownership between the reviewer(s) and the author, resulting in clean, bug-free code understood by multiple team members.

Monitoring

Another important aspect of our CICD pipeline is monitoring. To ensure that our applications consistently run as expected and that any change-related errors are detected and responded to immediately, each microapp had relevant monitors in place. To do this, we ensured all of our microapps were configured with proper logging, allowing us not only to detect and identify issues as they occurred, but also to collect data to analyze the performance of the application over time.

Specifically, since our API is the gateway for our frontend to interact with all of our microapps, it has a custom logging and monitoring utility that logs detailed information about each request to the API, including the request body, response body, and the time it took to process the request. Moreover, our proxy service logs all requests to each of the different microapps in the stack, allowing us to not only monitor individual requests, but to understand entire user flows and how students were interacting with their individual grade data on a platform level.

Beyond our detailed logging metrics, our monitors are also configured in Google Cloud to alert us when certain metrics were outside of expected ranges. For example, in the event that the average response time of the API exceeds one second, the dedicated on-call team are

notified via email, Slack, and a pager alert though the Google Cloud application. Similarly, if the rate of failed requests to the API exceeds one percent of total requests, the on-call team is similarly notified.

Chapter 6

Results

In order to evaluate the effectiveness of *GradeSuite*, we deployed the suite in an introductory computer science course (CS10) mid-semester in Fall 2024 and at the beginning of the semester in Spring 2025 at UC Berkeley where *GradeSuite* was made available to 170 and 48 students respectively. At the end of the semester, we also distributed an optional survey (section B.1) to elicit qualitative feedback. Throughout the deployment, we used our logging and analytics metrics to gather usage data from all enrolled students. Simultaneously, we continuously collaborated with course staff and students to ensure the suite met their needs.

6.1 Methodology

When launching *GradeSuite* in the Fall 2024 semester, we progressively rolled the suite out to students over a three week period starting on November 3rd, when it was released to the

first set of 20 students. The subsequent week, it was released to another 40 students, for a total of 60 students with access to *GradeSuite*. Finally, two weeks after the initial launch, it was released to the remaining 110 students in the course. The reason for this gradual rollout was to catch any early issues with the platform and allow instructors time to adjust.

In the fall, we analyzed application usage over a 52-day period from the initial release of *GradeSuite* on November 3rd to one week beyond the end of the semester on December 25th, when we stopped supporting student accesses. In the spring, *GradeSuite* was first released on March 12th, and our final analysis was conducted on April 30th, for a total term of 49 days.

For our end-of-semester qualitative survey of *GradeSuite*, we received responses from 59 students in the fall and 28 students in the spring. The survey contained open-ended, multiple choice, and Likert scale questions [31] that had options ranging from "Strongly disagree" to "Strongly agree." Each question was optional; students could skip questions they did not wish to answer by marking "N/A." We administered the survey using a Google Form sent to all students who had access to *GradeSuite*.

While writing the survey, we realized that students only interacted with *GradeSuite* through the *GradeView* and *Concept Map* modules. As such, in order to make the questions as intuitive for students as possible, we referred to *GradeView* as their entire student view of *GradeSuite* and *Concept Map* as the specific *Concept Map* module within *GradeView*'s user interface. However, it is worth noting that the student reflections of *GradeView* apply to the larger *GradeSuite* as a whole; for example, student-perceived grade update accuracy and

frequency during the spring semester when *GradeSync* was deployed reflects *GradeSync*'s efficacy.

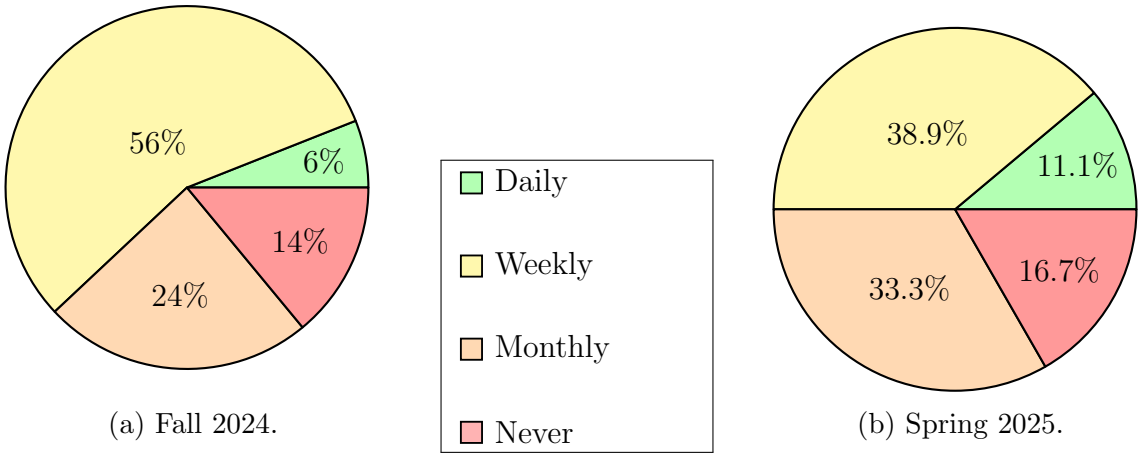
Another important note is that the survey was administered as part of a broader course survey, and as such, the incentives for completing it were set by the course staff. In the fall, there were no auxiliary incentives specified. In the spring, however, course staff offered extra credit applied to students' participation grade in the course. This meant that the demographic of students who completed the survey in the spring may disproportionately reflect a higher subset of students who did not regularly attend lecture, as those with good attendance stood less to gain from completing the survey.

6.2 Student Usage

One of the most important aspects of *GradeSuite* is the ability to provide students with conceptual feedback on their mastery of course material. To evaluate the effectiveness of *GradeSuite*, we looked at the usage data from each semester it was deployed as well as the post-semester surveys to inform us of how the students were interacting with and evaluating their mastery in the course through both *GradeView* and *Concept Map* respectively.

GradeView Usage

Over the fall semester, students logged into *GradeView* 3,231 times, for average of 19 uses per student. They collectively visited the *Grades* page to monitor overall course progress a total



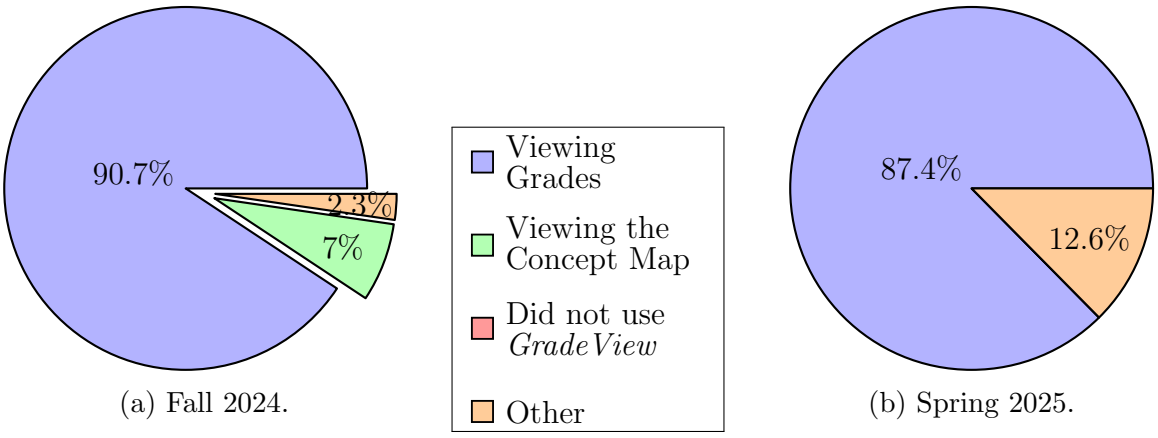
(a) Fall 2024.

(b) Spring 2025.

Figure 6.1: Normalized Student Responses to: "How often do you use *GradeView*?"

of 2,317 times, translating to an average of 13.6 views per student. In the spring semester, we logged a total of 867 accesses, averaging 18.1 uses per student. While these averages seem quite promising, we found that the distribution of students using *GradeView* was not as equally distributed as we had initially hoped. The survey usage results, shown in Figure 6.1, indicate that the vast majority of students decided to consistently use *GradeView*, with the average student across both semesters tending to use it every week.

Renormalizing the usage data to exclude the students who did not use *GradeView* as per Figure 6.1, we see that the remaining 86% students in the fall used *GradeView* 22.1 times on average, while the remaining 83.3% students in the spring used *GradeView* an average of 21.7 times. This seems to indicate that those who used *GradeView* tended to like it and continue using it throughout the semester to track their progress. While overall usage data are helpful to better understand how students interact with *GradeSuite*, we constructed our



(a) Fall 2024. (b) Spring 2025.

Figure 6.2: Normalized Student Responses to: "What do you primarily use *GradeView* for?"

survey to paint a more nuanced picture of the students' experiences with *GradeView* and its microservices such as *Concept Map*. Notably, we specifically asked students which services within *GradeSuite* they used as described by Figure 6.2.

Unsurprisingly, we found that the majority of students who used *GradeSuite* reported that they mainly used *GradeView* to view their grades in the class. This is consistent with our expectations, as *GradeView* is the primary tool, and therefore default landing page upon logging in, that students use to view their exact grades. Interestingly, we observed a 3.3% decrease in *GradeView* usage from Fall 2024 (Figure 6.2a) to Spring 2025 (Figure 6.2b), with only 87.4% of students reporting that they used *GradeView* to view their grades in the spring. We believe that the reason for this decrease may be due to several factors. The most notable source of variance between the two metrics is the number of responses in the spring, with the end-of-semester survey receiving only 28 responses, likely due to the smaller class size. Additionally, since the extra credit incentive offered in the spring disproportionately

benefitted students with imperfect attendance, we believe that those who did respond may reflect a subset of students who were less likely to know about *GradeView*, as the main method for informing students about the service was through lecture announcements.

Another key performance indicator we sought to measure during the early-stage deployment was the number of students who used *GradeSuite* to guide their retake decisions. In the pilot course, multiple exams concerning a variety of concepts were offered; their final grade per-topic was calculated as the maximum score they achieved on the topic across all offerings. This policy meant that students who had already demonstrated mastery of a given concept did not need to retake that question, allowing them to focus their efforts on the questions they had not yet demonstrated complete mastery on. For this reason, it was important for students to understand what concepts they had preformed poorly on to best guide their studying and retake decisions.

To explore how students used *GradeSuite* to optimize their retake decisions, we asked students how many retakes they had completed and how many they planned to complete but decided ultimately not to based on *GradeView*. We found that just over 72% of respondents who used *GradeView* in either semester reported that they had used it to guide their retake decisions, as shown in Figure 6.3. Additionally, Figure 6.4 depicts that, similarly across the semesters, 39.5% of respondents who used *GradeView* reported that they had used it to guide their decision not to retake at least one assignment that they originally planned to. While it may seem counterintuitive that students deciding not to retake an assignment or question aligns with mastery learning, it is important to note that their informed decision to abstain

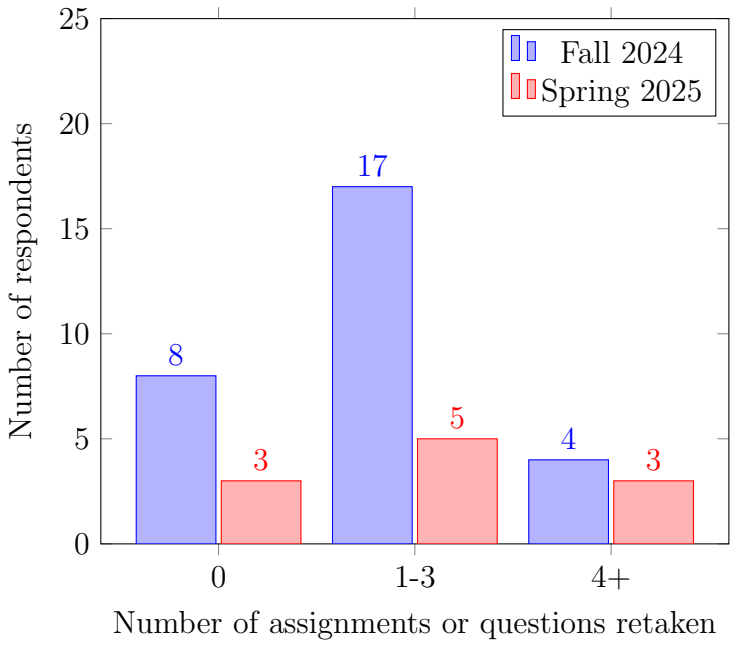


Figure 6.3: Student Responses to: "How many questions/assignments have you retaken based on scores shown in *GradeView*?"

reflects how their perceived mastery of the material was not aligned with their demonstrated mastery. For this reason, their decision not to go through with a retake validates *GradeView*'s ability to provide students with a more accurate picture of their individual course mastery.

Overall, the data demonstrate that *GradeView* served as an effective platform for grade monitoring and mastery assessment, with the majority of students actively engaging with the system during each semester. The platform's impact on student decision-making was particularly notable, with the vast majority of users leveraging *GradeView* to inform their retake choices and just short of half of its users applying its analytics it to make more informed decisions about forgoing initially planned retakes. While deployment timing var-

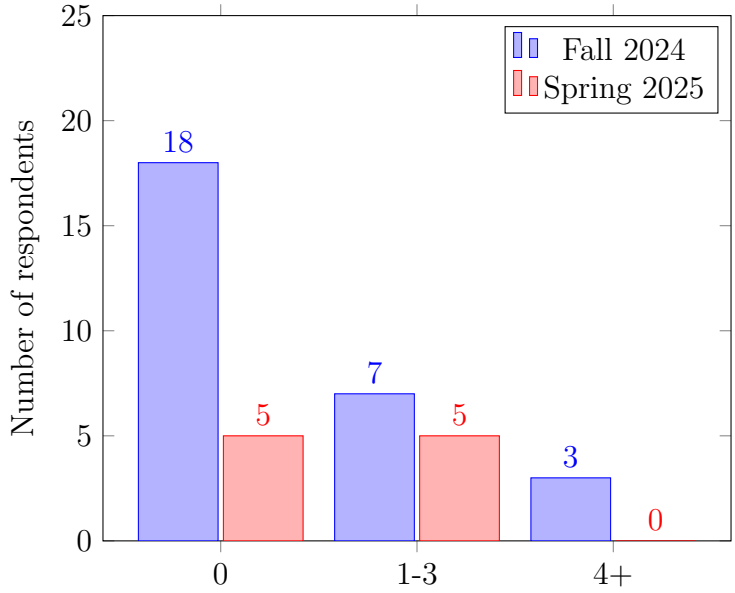
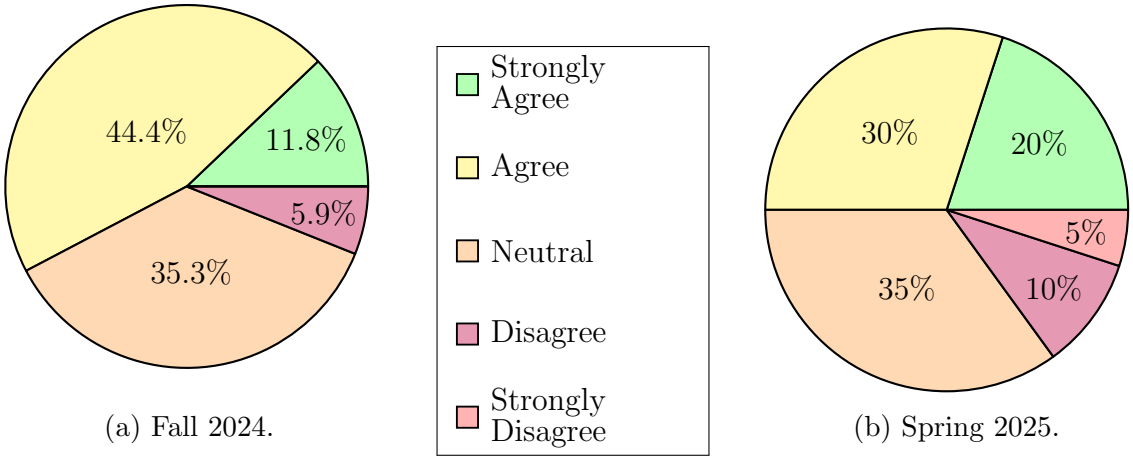


Figure 6.4: Student Responses to: "How many questions/assignments did you anticipate retaking, but chose not to based on scores shown in *GradeView*?"

ied between semesters, the consistent usage patterns and positive survey responses suggest that *GradeView* successfully fulfilled its core objective of providing actionable feedback for mastery-based learning decisions.

***Concept Map* Usage**

After launching in the fall, students used *Concept Map* 627 times (3.6 per student, on average). The following semester, students viewed the microapp a total of 523 times (10.9 per student, on average) for a dramatic average usage increase of over 300%. Of those who did use *Concept Map*, over 55% in the fall and 50% in the spring (Figure 6.5) indicated



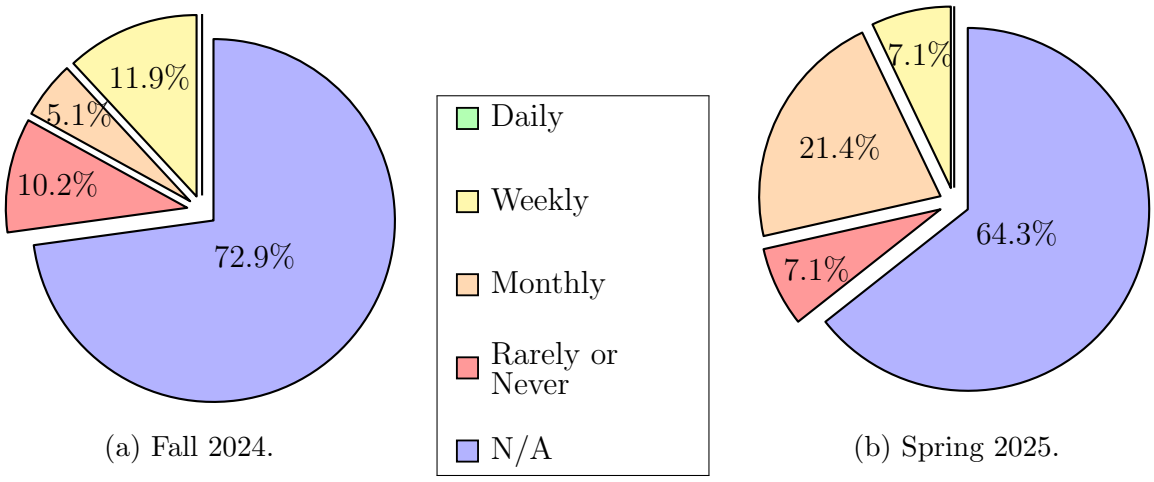
(a) Fall 2024.

(b) Spring 2025.

Figure 6.5: Normalized Student Responses to: "*Concept Map* helps me track my progress in the class."

that it helped them track their progress in the class, meaning that the module was useful in helping students identify their own conceptual gaps. Unfortunately, we also noticed that 6% of the same sample set in the fall and 15% in the spring felt as though *Concept Map* was explicitly not helpful for tracking their progress. This is a clear indication that future work is needed to improve the *Concept Map* module to better serve the needs of students.

One surprising result we noticed in Figure 6.2 is that the overall proportion of students who primarily used *Concept Map* was substantially lower than we had expected in both semesters. Despite the increase in recurring (daily, weekly, or monthly) usage in the spring as demonstrated by Figure 6.6 as well as the per-student average usage increase, the number of students who used *Concept Map* as their primary *GradeSuite* service dropped from only 7% in the fall to a staggering 0% in the spring. This was a stark contrast to our expectations,



(a) Fall 2024. (b) Spring 2025.

Figure 6.6: Student Responses to: "How often do you use *Concept Map*?"

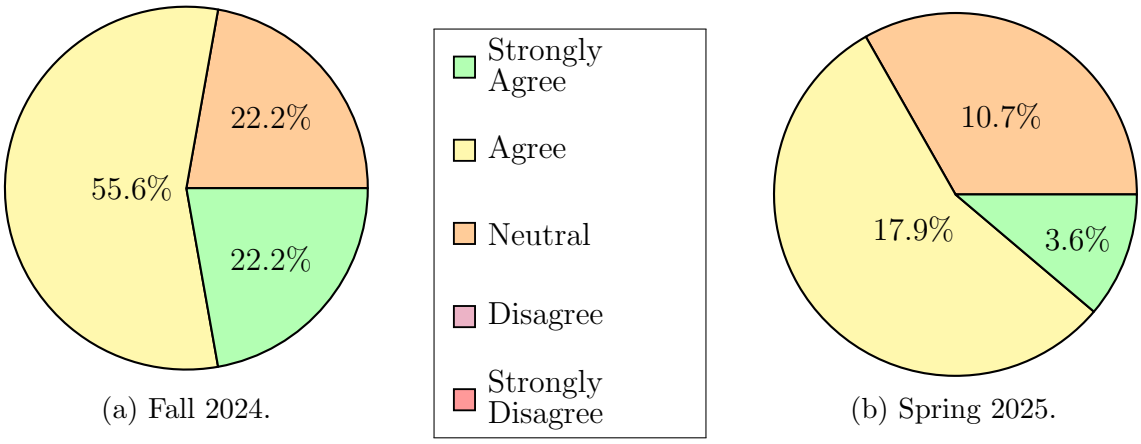
as we had expected the primary usage of *Concept Map* to increase markedly in the spring.

Initially, we expected the primary usage statistics in the fall to be relatively low. This is because the *Concept Map* module was launched quite early in its development, while it was not yet fully polished, meaning that, for some students, it did not depict the correct mastery information and progression. Specifically, throughout the duration of the semester, all nodes were shown (not just those that had been covered in the class), and the edges were not colored based on the course's progression as we had initially designed (although this discrepancy was not advertised to students). Moreover, for the vast majority of the semester, student mastery was not updated in real-time since we had not yet integrated with *GradeSync*, translating to the *Concept Map* module inconsistently reflecting the student's up-to-date mastery while waiting for course staff to manually update the data. Because of these early-stage issues, we

believe that students may have been less inclined to use *Concept Map* and more inclined to use *GradeView*, where the data was generally accurate across the semester. This is consistent with the results in Figure 6.2a, in which 90.7% of students who used the platform indicated that *GradeView* was their primary *GradeSuite* service.

Given our expectations of lower *Concept Map* use in the fall, we anticipated a marked increase in the spring. We hypothesized that, after working out the kinks discovered while launching *Concept Map* in the fall such as its incorrect mastery and course progression information, students would be more inclined to use *Concept Map*. However, our survey results in Figure 6.6 do not indicate as substantial of an increase as we had expected. We believe that this is due to a variation in *how* students used *Concept Map* between the two pilot semesters. Namely, we think that students in the fall used *Concept Map* to better understand the course's structure and progression, while students in the spring referenced *Concept Map* to identify gaps in their own mastery. This is evidenced by the fact that, in the fall, the entire concept map was viewable, despite representing incorrect mastery information. In the spring, however, only the nodes that had been covered in the class were rendered, meaning that students were not able to preview upcoming content with *Concept Map*.

Another reason we initially hypothesized could be the source of lower primary usage was because students may have been confused about how to use *Concept Map*. We were concerned that the user interface may have been too confusing for students. Specifically, while early iterations of *Concept Map* leveraged edge coloring to indicate whether concepts



(a) Fall 2024. (b) Spring 2025.

Figure 6.7: Normalized Student Responses to: "I understood how to interpret the data presented in *Concept Map*."

had been taught or not taught, the *Concept Map* module during the course's offering did not support this feature despite a respective legend being shown. Given our concern, we included a question in our end-of-semester survey to evaluate how comfortable students were with reading their mastery from *Concept Map*. To our surprise, the survey results in Figure 6.7 contradicted our hypothesis, clearly indicating that no students struggled to interpret their mastery information in *Concept Map*. While we had initially planned to revise the *Concept Map* UI to further increase accessability beyond removing the deprecated legend, the survey results motivated our decision to discard this project.

In the future, we believe it may be beneficial to revert back to the original *Concept Map* design, rendering the entire course's conceptual breakdown, but with edges colored based on the course's progression rather than generating the map as the course progresses. This would allow students to continue using *Concept Map* to preview upcoming content, while

still being able to view their own mastery information.

Student Feedback

In addition to the post-semester survey where students were asked to provide their own, open-ended feedback on *GradeView* and *Concept Map* directly, we also provided a separate survey (section B.2) for students to complete throughout the semester to detail the bigger picture of students' experiences with the suite. Many of the responses from these surveys in the fall regarded grading inaccuracies and the need for more frequent grade updates by course staff. These comments were largely unsurprising; early implementations of *GradeSync* were not fully supported resulting in extended delays as course staff had to manually compute and update grades. Moreover, bugs introduced by early versions of *GradeSync* in addition to course staff manually updating the Google Sheet's format resulted in misformed data which propagated to *GradeView* and *Concept Map* and similarly affected student mastery information.

In contrast, the spring survey responses were overwhelmingly positive, with students reporting that they found *GradeView* and *Concept Map* helpful in identifying gaps in their own mastery and in visualizing the course's progression. This highlights the need for *GradeSync*'s automated grade synchronization which would not only provide real-time grade updates, but also reduce the amount of miscalculations reported by students. These reports likewise validate the proposed benefits of *GradeView* for both students and course staff, as such errors

would not have otherwise been caught if students were not able to view their course data.

Student comments relating to the platform itself, however, largely touted the value of *GradeSuite* across both semesters. Many students reported that *GradeView* was helpful in viewing their overall scores and the areas where they were struggling. Similarly, multiple students mentioned that it was nice to have all of their course data in one place. *Concept Map*, while not used as frequently, was still viewed positively as students found it helpful in visualizing the course's progression and in identifying the topics that would be tested in exams, allowing them to better prepare. These comments are consistent with our evaluation of Figure 6.6, indicating that students preferred the ability to preview upcoming content with *Concept Map*.

One critique we received, though, was that certain students who had changed their emails since the beginning of the semester were not able to access the dashboard with their new email. This is because the underlying Google Sheet responsible for storing their course data still reflected their original email. As such, one future direction of the *Instructor Dashboard* project may be to allow instructors to manually update student emails. Alternatively, it may be beneficial to integrate *GradeSync* with the university's registration systems to automate this process.

6.3 Course Staff Impact

While the primary focus of *GradeSuite* is to provide students with real-time feedback on their mastery of course material, we also sought to measure the impact of *GradeSuite* on the course staff as they used *GradeSuite* to manage grades and view student mastery progression. To do this, we conducted interviews with the course staff to gather their feedback on the suite and its impact on the course.

Our interviews indicated that *GradeSuite* significantly improved operational efficiency, accuracy, and pedagogical insights. Specifically, one of our interviews unveiled that calculating a student's concept mastery in real time during office hours previously required at least ten minutes per student, cutting into valuable TA support time. With the introduction of *Instructor Dashboard*, staff gained access to itemized reports and respective visualizations of individual student mastery, enabling more targeted and timely assistance.

Reduction of TA Time Invested

Prior to their integration with *GradeSuite*, CS10 course staff invested substantial amounts of time in managing the course's complex grading infrastructure. Head TAs spent an estimated 10 hours per week performing manual grade management tasks, including: exporting and consolidating grades from multiple learning platforms (Gradescope, PrairieLearn, and iClicker), processing retake submissions, calculating concept mastery scores, managing late submissions, and updating the status of students with Incomplete grades from prior

semesters. These tasks were not only time-consuming but also prone to human error due to the manual nature of data handling across multiple spreadsheets and platforms.

CS10's use of the *GradeSync* microapp transformed their grading workflow through its data-management platform and various automations. The system's nightly synchronization scripts automatically handle grade consolidation, retake processing, and mastery calculations, reducing the related weekly administrative workload to just one hour. The remaining time is primarily spent on high-value tasks such as reviewing student-flagged discrepancies which itself was largely eliminated by *GradeSuite*. This represents a 90% reduction in time spent on grade processing and related logistical communication. Moreover, the 13 cases of final grade discrepancies that were reported the semester prior to *GradeSync*'s integration reduced to zero the following semester.

Additionally, the automation of grade updates significantly reduced the time spent responding to student inquiries about grades and retake status. Previously, TAs would need to manually verify grade calculations and retake eligibility for each student inquiry, a process that could take several minutes per request. With *GradeSuite* providing real-time grade updates, many of these inquiries were eliminated entirely, as students could access this information directly through the platform.

The substantial reduction in administrative overhead afforded by *GradeSuite* allowed course staff to redirect their time to more valuable pedagogical activities, such as developing improved course materials, providing more detailed feedback on assignments, and spending more quality time with students during office hours. Course staff reported that this shift not

only improved their job satisfaction but also enhanced the overall quality of instruction and student support they were able to provide.

Frequency of Grade Updates

The frequency and timeliness of official grade updates also improved substantially. Historically, final grade recalculations were performed in a single end-of-term batch, increasing the risk of overlooking late retakes or corrections. In contrast, *GradeSync* introduced hourly synchronization from Gradescope and daily synchronization with PrairieLearn, resulting in near real-time grade accuracy and more timely feedback for students. Staff noted that this enhanced alignment enabled quicker responses to students requesting deadline extensions. Likewise, the introduction of *GradeView* and *Concept Map* as real-time indicators of mastery assisted course staff in identifying students who may be struggling with the material and in need of additional support without having to manually recalculate a student's respective mastery.

During the Fall 2024 semester, we integrated *GradeSync* with *GradeView* later on in the semester, meaning that students were not able to leverage the continuously up-to-date data provided by *GradeSync* for the majority of the fall semester. As mentioned in section 6.2, students' most common critique was *GradeView* needed to be updated more often. Since adopting *GradeSync* into our workflow for the duration of the spring semester, it is unsurprising that we did not receive any such reports on our surveys.

GradeSuite's effectiveness in reducing final grade calculation errors, combined with the increased frequency of grade updates, and dramatic reduction of staffing time suggests that it can effectively support more flexible course policies — such as frequent retakes and extended deadlines — without compromising accuracy or increasing administrative burden on instructional staff. The system's design that supports arbitrarily complex grade calculations as defined by course staff in the Google Sheets interface has proven particularly valuable in the context of mastery learning, giving instructors the freedom to design *any* equitable grading policy they want. By automating the processing of these intricate calculations while maintaining consistent data across all platforms, *GradeSync* has effectively eliminated a significant source of stress and inefficiency in course administration, simultaneously ensuring that students receive accurate and timely feedback on their academic performance.

Chapter 7

Future Work

The successful implementation of *GradeSuite* in CS10 has demonstrated its potential to enhance student success. Moving forward, we plan to expand *GradeSuite* to both lower and upper division Computer Science courses at UC Berkeley, further refining the system's effectiveness across different instructional settings. Our development roadmap prioritizes continued development of additional microapps to expand the *GradeSuite* ecosystem, such as *AutoRemind*, an intelligent notification system that engages students through their preferred digital platforms. Additionally, we plan to reconfigure *GradeView* to better support multiple semesters, without the need to re-deploy multiple instances of the microapp. We would also like to revise our cloud infrastructure management, reducing the amount of manual configuration necessary by using IAC to declaratively describe the desired state of the system. Finally, we hope to keep analyzing student and staff usage patterns while conducting further studies to develop a more comprehensive understanding of how best support

continued adoption of mastery learning policies through the *GradeSuite* project.

7.1 *AutoRemind*

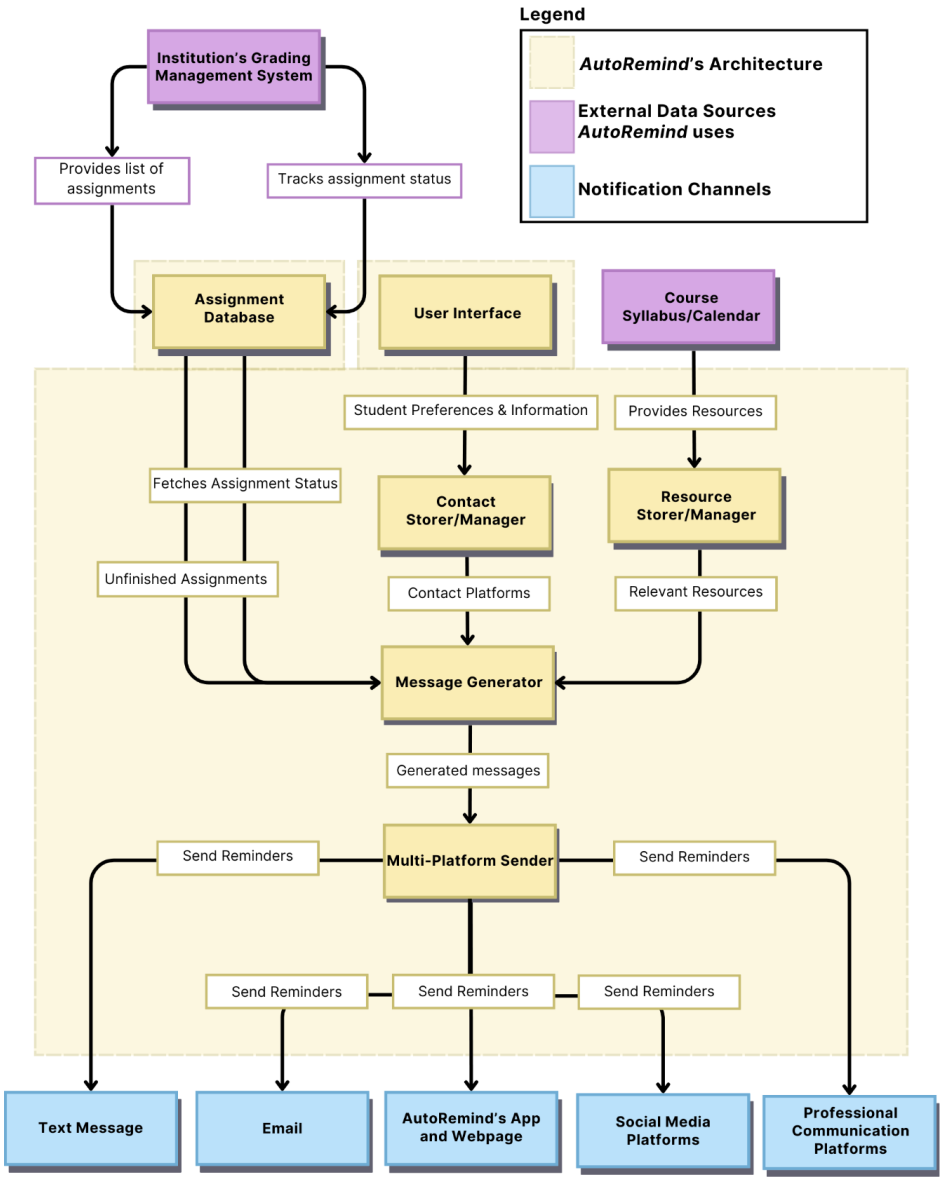


Figure 7.1: *AutoRemind* Architecture and Data Flow.

One new microapp we have been working on developing over the past year is *AutoRemind* [11], which provides students with dynamic reminders of their course work throughout the semester on nonconventional platforms such as Discord and Instagram. The key incentive for *AutoRemind* is that early implementations of progressive deadline policies in support of mastery learning, specifically deadline abstraction, have seen challenges as students fall behind, intending to catch back up later in the semester [34]. Ultimately, early studies found that students commonly became overwhelmed and eventually decided to give up. *AutoRemind* mitigates this by tracking each student's progress throughout the semester, sending them reminders on a variety of student-configurable platforms of the outstanding assignments they need to complete alongside relevant course materials they may benefit from reviewing.

The general architecture and flow of *AutoRemind* is shown in Figure 7.1. *AutoRemind* integrates with *GradeSync* to keep track of each students' progress, and scrapes the course's syllabus and website for key content such as Office Hours times, assignment due dates, and other course-specific information. *AutoRemind* then uses a custom Retrieval-Augmented Generation (RAG) agent to parse the relevant information and send it to the students as reminders on the platforms they have configured. While *AutoRemind* is currently limited to email, Discord, and SMS, we intend to expand its reach to include more platforms in the future. Right now, *AutoRemind* is still being actively integrated with *GradeView*, but we expect it to be an invaluable addition to *GradeSuite*.

7.2 Grade Projections

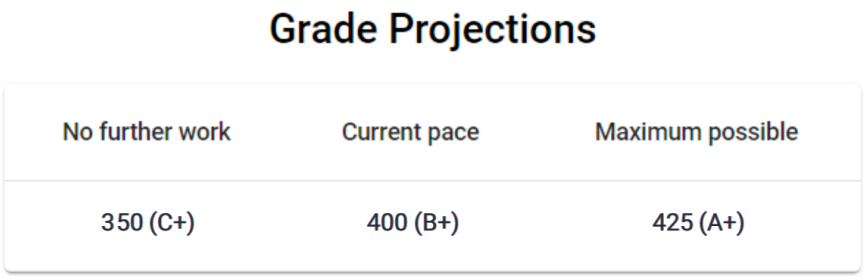


Figure 7.2: Initial Implementation of Grade Projections in *GradeView*.

One major project we worked on during the semester was adding grade projections to *GradeView* as shown in Figure 7.2. The grade projections would allow students to see their trajectory in the course and extrapolate from it how they might perform in the future. While these projects were initially successful, we later realized that they did not naturally support grade clobbering [24]; the *Maximum possible* projection was based only on assignments the student had not yet completed, not factoring in those that students could later clobber through the course's retake policy. Additionally, we were concerned that students who fell behind or were initially struggling with the course content may be discouraged by these projections. As such, we decided to remove them from *GradeView* prior to releasing it in the Fall 2024 semester.

Despite removing the grade projections from *GradeView*, we believe that this could be an extremely useful feature for students to have. We think that there could be a graceful way to add them back without the negative side effects initially identified. In the future,

we would like to explore such integrations as adding topic-centric projections that track the trajectory of each concept itself, rather than the overall course grade. Moreover, projecting concept-level mastery is more consistent with the mastery-based focus of the *GradeSuite* project.

7.3 *GradeView* Multi-Environment Support

While ideating how to improve course systems to better support mastery learning, we realized that properly managing incomplete students, as we have previously defined, is an extensive burden due to its lack of support from traditional LMSs. *GradeView* presently supports such functionality, but the process to do so is cumbersome and expensive, requiring deployment of additional instances for each course instance, each with their own Google Sheet integration to provide access control and data isolation. While this approach is both effective and viable, it is not as scalable as we had initially hoped; configuring each new instance of *GradeView* takes time and introduces additional processing overhead, resulting in increased cloud or energy costs depending on the deployment strategy.

To address these issues, we would like to add multi-environment support directly to *GradeView*. Specifically, we hope to configure the *GradeView* API to include 'course' and 'semester' based resources, so that API calls could be made to retrieve data for a given course and semester. To support the new API specification, we will need to similarly refactor *GradeView*'s frontend microservice to make the appropriate API calls for the logged-in user. For

example, instead of *GradeView* requesting `/api/v2/assignments` to get a list of all assignments, it would make a request to `/api/v2/courses/cs10/semesters/fa24/assignments` which would fetch all CS10 assignments specifically for the Fall 2024 semester. This revised strategy would allow us to deploy a single instance of *GradeView* for an institution, while still allowing each course to have their own isolated access control and grade data per semester.

For proper support, we will similarly need to update the structuring of the Redis caching layer to include course- and semester-based classification. Moreover, we will need to configure our cron jobs to pull data from all active semesters' Google Sheets and populate it in the Redis cache accordingly. We expect that the additional economic overhead of running the additional cron jobs and managing the increased Redis storage will minimally affect operational costs, especially when compared to deploying a new instance of *GradeView* for each course instance.

7.4 Infrastructure as Code (IAC)

One direction we wish to move in (or rather back towards) is in declaratively managing our infrastructure as code using a tool such as Terraform [51]. This would be extremely beneficial to the *GradeSuite* project because each of the different microapps' microservices are currently configured manually, meaning inconsistencies or mistakes when updating infrastructure may not be caught or directly documented. Moreover, the implicitly versioned nature of IAC through our Git repository would allow us to more easily track and deploy changes to the

infrastructure in the specific repositories that it related to.

7.5 Future Studies

As we continue to expand *GradeSuite*, we feel it is of paramount importance to continue evaluating its efficacy in supporting mastery learning policies. We plan to continue our work in this area by conducting further studies on the impact of *GradeSuite* on student success as well as its ability to assist instructors in their ability to monitor concept mastery development and make immediate adjustments to their curriculum during the semester to ensure topics are thoroughly understood before advancing to more complex material.

Additionally, we would like to establish a more standardized approach to collecting qualitative feedback from students by developing and administering our own, dedicated end-of-semester surveys. Our previous survey efforts revealed significant variations in response rates and potential sampling biases between semesters, particularly due to differing incentive structures. To address these limitations, we aim to implement a uniform survey methodology across all future analyses of *GradeSuite*, controlling for potential confounding variables in our data collection process.

Chapter 8

Conclusion

The development and deployment of *GradeSuite* represents a significant advancement in supporting mastery learning within traditional educational environments. Through its innovative microservice architecture and seamless integration with existing Learning Management Systems, *GradeSuite* has demonstrated its effectiveness in both enhancing student learning outcomes and reducing administrative overhead for course staff. The successful implementation in CS10 at UC Berkeley has provided valuable insights into the platform's impact and potential for broader adoption.

The architectural foundation of *GradeSuite* has been designed to minimize barriers to adopting mastery learning policies in existing courses such as CS10, where lack of LMS support proved obstructed the adoption of principle mastery learning policies. Through its various microapps, the platform integrates seamlessly with existing course infrastructure while providing the flexibility needed to implement diverse grading policies. The system's

evolution through various deployment strategies — from cloud to bare-metal and back — demonstrates its adaptability to different institutional environments. Particularly notable is the platform's use of Google Sheets as a familiar interface for instructors, enhanced by Redis caching for scalability, which allows course staff to implement complex mastery-based policies without requiring significant technical expertise or infrastructure changes. This careful balance between technical capability and ease of adoption has enabled instructors to focus on crafting effective mastery learning policies unrestricted by the technical limitations of their LMS.

The deployment of *GradeSuite* has demonstrated several transformative effects in our pilot class. The platform's ability to provide up-to-date, concept-level mastery data has enabled students to identify and address learning gaps more effectively, proving particularly valuable for targeted studying and identifying retake opportunities. By supporting flexible deadlines and post-term completion through automated grade synchronization, *GradeSuite* has additionally made mastery learning more accessible to students who may require extra time to achieve and demonstrate proficiency.

The organizational structure supporting *GradeSuite*'s development has been equally crucial to its success. What began as an individual initiative quickly accrued a team four developers, and has since grown into a robust team of over twenty contributors across its various microapps, managed through a carefully designed hierarchical structure that promotes both clear accountability and open communication.

Looking forward, several promising directions for *GradeSuite*'s evolution have emerged.

The planned integration of *AutoRemind* will enhance student engagement through personalized reminders and progress tracking across various communication platforms. Infrastructure improvements, including the implementation of IAC and multi-environment support for *GradeView*, will improve scalability and reduce operational overhead. Additionally, continued analysis of student usage patterns and learning outcomes will provide valuable insights into the effectiveness of real-time measures of mastery learning.

The implications of *GradeSuite* extend far beyond UC Berkeley, offering a blueprint for institutions facing similar challenges in implementing mastery learning. Its success demonstrates that technical limitations of existing LMSs need not prevent the adoption of mastery-based pedagogical approaches. The platform's integration-first architecture, which prioritizes working with existing tools rather than replacing them, provides a practical model for educational technology development that minimizes adoption barriers. As more institutions seek to implement mastery learning policies, *GradeSuite* presents a compelling example of how to bridge the gap between pedagogical aspirations and technical constraints.

In conclusion, *GradeSuite* represents a significant step forward in tooling to support mastery learning. Its successful deployment has demonstrated the feasibility of implementing flexible, student-centered learning approaches at scale while reducing administrative overhead. As the platform continues to evolve and expand to other courses, it will hopefully facilitate the adoption of equitable grading practices and, with it, student success.

Bibliography

[1] *Assessment Reports: Evaluating Student Mastery Results (AMP)*. Schoology Learning. URL: https://uc.powerschool-docs.com/en/schoology/latest/assessment-reports-evaluating-student-results-by-i (visited on 05/12/2025).

[2] Connor Bernard. *AFA-Tooling*. GitHub. URL: https://github.com/AFA-Tooling (visited on 05/13/2025).

[3] Connor Robert Bernard et al. "Supporting Mastery Learning Through an Adaptive Grade Portal". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, Mar. 15, 2024, pp. 1568–1569. ISBN: 979-8-4007-0424-6. DOI: 10.1145/3626253.3635621. URL: https://doi.org/10.1145/3626253.3635621 (visited on 03/17/2025).

[4] Manan Bhargava et al. "GradeSync: A Tool for Automating Incomplete Processing to Support Mastery Learning". In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSETS 2025. New York, NY, USA: Association

for Computing Machinery, Feb. 18, 2025, pp. 1387–1388. ISBN: 979-8-4007-0532-8. DOI: 10.1145/3641555.3705192. URL: https://dl.acm.org/doi/10.1145/3641555.3705192 (visited on 03/17/2025).

[5] Benjamin S. Bloom. "Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1." In: *Evaluation Comment* 1.2 (May 1968). URL: https://eric.ed.gov/?id=eD053419 (visited on 03/20/2025).

[6] Vaughn Malcolm Bradley. "Learning Management System (LMS) Use with Online Instruction". In: *International Journal of Technology in Education* 4.1 (2021), pp. 68–92. URL: https://eric.ed.gov/?id=EJ1286531 (visited on 03/18/2025).

[7] *Build Software Better, Together*. GitHub. URL: https://github.com (visited on 04/30/2025).

[8] Vasa Buraphadeja and Vilasinee Srisarkun. "Mastery Learning in CS1: A Longitudinal Study during and Post-Pandemic". In: *Discover Education* 3.1 (Dec. 2, 2024), p. 260. ISSN: 2731-5525. DOI: 10.1007/s44217-024-00361-x. URL: https://link.springer.com/10.1007/s44217-024-00361-x (visited on 03/18/2025).

[9] Alberto Cañas et al. "CmapTools: A Knowledge Modeling and Sharing Environment". In: Concept Maps: Theory, Methodology, Technology Proceedings of the First International Conference on Concept Mapping. Sept. 2004.

[10] *Canvas*. Instructure. URL: https://www.instructure.com/canvas (visited on 05/13/2025).

[11] Oindree Chatterjee et al. "AutoRemind: Improving Student Academic Performance Through a Personalized and Automated Notification System". In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, Feb. 18, 2025, pp. 1411–1412. ISBN: 979-8-4007-0532-8. DOI: 10.1145/3641555.3705133. URL: https://doi.org/10.1145/3641555.3705133 (visited on 04/29/2025).

[12] Se-Hang Cheong, Yain-Whar Si, and Raymond K. Wong. "Online Force-Directed Algorithms for Visualization of Dynamic Graphs". In: *Information Sciences* 556 (May 1, 2021), pp. 223–255. ISSN: 0020-0255. DOI: 10.1016/j.ins.2020.12.069. URL: https://www.sciencedirect.com/science/article/pii/S0020025520312354 (visited on 05/01/2025).

[13] *Competency-Based Education*. D2L. Feb. 1, 2023. URL: https://www.d2l.com/solutions/higher-education/competency-based-education (visited on 05/12/2025).

[14] *Crontab(5) - Linux Manual Page*. URL: https://man7.org/linux/man-pages/man5/crontab.5.html (visited on 05/13/2025).

[15] *D3 by Observable — The JavaScript Library for Bespoke Data Visualization*. URL: https://d3js.org (visited on 05/01/2025).

[16] *Data and IT Resource Classification Standard — Information Security Office*. URL: https://security.berkeley.edu/data-classification-standard (visited on 05/01/2025).

[17] *Docker: Accelerated Container Application Development*. May 10, 2022. URL: https://www.docker.com (visited on 03/18/2025).

[18] Stephen H. Edwards et al. "Developing a Playbook of Equitable Grading Practices". In: *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 2*. SIGCSE Virtual 2024. New York, NY, USA: Association for Computing Machinery, Dec. 5, 2024, pp. 283–284. ISBN: 979-8-4007-0604-2. DOI: 10.1145/3649409.3691071. URL: https://doi.org/10.1145/3649409.3691071 (visited on 05/12/2025).

[19] John Ellson et al. "Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools". In: *Graph Drawing Software*. Ed. by Michael Jünger and Petra Mutzel. Red. by Gerald Farin et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 127–148. ISBN: 978-3-642-18638-7. DOI: 10.1007/978-3-642-18638-7_6. URL: http://link.springer.com/10.1007/978-3-642-18638-7_6 (visited on 03/18/2025).

[20] *Flask*. 2010. URL: https://flask.palletsprojects.com/en/stable (visited on 03/18/2025).

[21] OpenJS Foundation. *Node.Js — Run JavaScript Everywhere*. 2025. URL: https://nodejs.org/en (visited on 03/25/2025).

[22] Koen B. Franken. "Promoting Mastery Goal Orientation by Developing and Implementing a Mastery Focused Canvas Dashboard". MA thesis. Eindhoven, Netherlands: Eindhoven University of Technology, Aug. 31, 2023. 80 pp.

[23] Dan Garcia et al. "A's for All (As Time and Interest Allow)". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023: The 54th ACM Technical Symposium on Computer Science Education. Toronto ON Canada: ACM, Mar. 2, 2023, pp. 1042–1048. ISBN: 978-1-4503-9431-4. DOI: 10.1145/3545945.3569847. URL: https://dl.acm.org/doi/10.1145/3545945.3569847 (visited on 05/12/2025).

[24] Dan Garcia et al. "Equitable Grading Best Practices". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2023: The 54th ACM Technical Symposium on Computer Science Education. Toronto ON Canada: ACM, Mar. 2023, pp. 1200–1201. ISBN: 978-1-4503-9433-8. DOI: 10.1145/3545947.3569602. URL: https://dl.acm.org/doi/10.1145/3545947.3569602 (visited on 05/13/2025).

[25] James Garner, Paul Denny, and Andrew Luxton-Reilly. "Mastery Learning in Computer Science Education". In: *Proceedings of the Twenty-First Australasian Computing Education Conference*. ACE '19. New York, NY, USA: Association for Computing Machinery, Jan. 29, 2019, pp. 37–46. ISBN: 978-1-4503-6622-9. DOI: 10.1145/

3286960.3286965. URL: https://doi.org/10.1145/3286960.3286965 (visited on 03/17/2025).

[26] *Git - Git-Blame Documentation*. URL: https://git-scm.com/docs/git-blame (visited on 05/13/2025).

[27] *Gradescope — Save Time Grading*. URL: https://www.gradescope.com (visited on 05/13/2025).

[28] *How Do I Use the Learning Mastery Gradebook to View Outcome Results in a Course?* Instructure Community. July 20, 2020. URL: https://community.canvaslms.com/t5/Instructor-Guide/How-do-I-use-the-Learning-Mastery-Gradebook-to-view-outcome/ta-p/775 (visited on 03/25/2025).

[29] *iClicker: Student Response & Classroom Engagement Tools*. iClicker. URL: https://www.iclicker.com (visited on 05/13/2025).

[30] *Jira — Issue & Project Tracking Software — Atlassian*. URL: https://www.atlassian.com/software/jira (visited on 04/17/2025).

[31] R. Likert. "A Technique for the Measurement of Attitudes." In: *Archives of Psychology* 22 140 (1932), pp. 55–55.

[32] *Linear – Plan and Build Products*. URL: https://linear.app (visited on 04/17/2025).

[33] Vedansh Malhotra and Dan Garcia. "Steering Student Behavior and Performance Toward Success with Mastery Learning through Policy Optimization". In: *Proceedings*

of the 2024 on ACM Virtual Global Computing Education Conference V. 1. SIGCSE Virtual 2024. New York, NY, USA: Association for Computing Machinery, Dec. 5, 2024, pp. 144–150. ISBN: 979-8-4007-0598-4. DOI: 10.1145/3649165.3690109. URL: https://dl.acm.org/doi/10.1145/3649165.3690109 (visited on 03/17/2025).

[34] Vedansh Malhotra and Dan Garcia. "The Effect of Messaging on Project Completion Rates in an Introductory Computing Class Utilizing Mastery Learning". In: *Proceedings of the ACM Conference on Global Computing Education Vol 2*. CompEd 2023. New York, NY, USA: Association for Computing Machinery, Dec. 5, 2023, p. 198. ISBN: 979-8-4007-0374-4. DOI: 10.1145/3617650.3624932. URL: https://doi.org/10.1145/3617650.3624932 (visited on 04/29/2025).

[35] Vedansh Malhotra, Jenny Mendez Mendez, and Daniel Garcia. "Mastery with Method: Calibrating Policies to Boost Completion and Sentiment in a Computing Course Using Mastery Learning". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, Mar. 15, 2024, pp. 1738–1739. ISBN: 979-8-4007-0424-6. DOI: 10.1145/3626253.3635629. URL: https://doi.org/10.1145/3626253.3635629 (visited on 05/12/2025).

[36] *Manifesto for Agile Software Development*. URL: https://agilemanifesto.org (visited on 04/17/2025).

[37] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", Oct. 18, 2011. 115 pp. ISBN: 978-1-4493-1990-8. Google Books: `eABpzyTcJNIC`.

[38] *Material UI: React Components That Implement Material Design*. URL: `https://mui.com/material-ui` (visited on 03/18/2025).

[39] *Matplotlib — Visualization with Python*. URL: `https://matplotlib.org` (visited on 05/02/2025).

[40] Brendan McCane et al. "Mastery Learning in Introductory Programming". In: *Proceedings of the Nineteenth Australasian Computing Education Conference*. ACE '17. New York, NY, USA: Association for Computing Machinery, Jan. 31, 2017, pp. 1–10. ISBN: 978-1-4503-4823-2. DOI: `10.1145/3013499.3013501`. URL: `https://dl.acm.org/doi/10.1145/3013499.3013501` (visited on 03/17/2025).

[41] *Nginx*. URL: `https://nginx.org` (visited on 03/18/2025).

[42] *OAuth 2.0 — OAuth*. URL: `https://oauth.net/2` (visited on 03/18/2025).

[43] *Pandas - Python Data Analysis Library*. URL: `https://pandas.pydata.org` (visited on 05/02/2025).

[44] *PrairieLearn*. URL: `https://www.prairielearn.com` (visited on 05/13/2025).

[45] Angelina Amalia Putri. "The Effectiveness of Khan Academy as a Science Learning Support to Improve Student's Mastery of Skills : Literature Review". In: *Journal of*

Environmental and Science Education 1.2 (2 Sept. 29, 2021), pp. 52–56. ISSN: 2775-2518. DOI: 10.15294/jese.v1i2.50370. URL: https://journal.unnes.ac.id/sju/jese/article/view/50370 (visited on 03/18/2025).

[46] Alex Rayón, Mariluz Guenaga, and Asier Núñez. "Integrating and Visualizing Learner and Social Data to Elicit Higher-Order Indicators in SCALA Dashboard". In: *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*. I-KNOW '14. New York, NY, USA: Association for Computing Machinery, Sept. 16, 2014, pp. 1–4. ISBN: 978-1-4503-2769-5. DOI: 10.1145/2637748.2638435. URL: https://doi.org/10.1145/2637748.2638435 (visited on 03/17/2025).

[47] *React*. URL: https://react.dev (visited on 03/18/2025).

[48] *Redis - The Real-time Data Platform*. Redis. URL: https://redis.io (visited on 03/18/2025).

[49] Mahammad Sharifov, Samaya Safikhanova, and Abdulsalam Mustafa. "Review of Prevailing Trends, Barriers and Future Perspectives of Learning Management Systems (LMSs) in Higher Institutions". In: *International Journal of Education and Development using Information and Communication Technology (IJEDICT)* 17.3 (2021), pp. 207–216. ISSN: EISSN-1814-0556. URL: https://files.eric.ed.gov/fulltext/EJ1335692.pdf.

[50] Ross Strader and Candace Thille. "The Open Learning Initiative: Enacting Instruction Online". In: *Gamer Changers: Education and Information Technologies*. Ed. by Diana Oblinger. Washington, D.C.: EDUCAUSE, 2012, pp. 201–213. ISBN: 978-1-933046-00-6.

[51] *Terraform — HashiCorp Developer*. Terraform — HashiCorp Developer. URL: `https://developer.hashicorp.com/terraform` (visited on 04/22/2025).

[52] Candace Thille. "MOOCs and Technology to Advance Learning and Learning Research Opening Statement: MOOCs and Technology to Advance Learning and Learning Research (Ubiquity Symposium)". In: *Ubiquity* 2014 (April Apr. 1, 2014), 1:1–1:7. DOI: `10.1145/2601337`. URL: `https://dl.acm.org/doi/10.1145/2601337` (visited on 03/17/2025).

[53] Jeramey Tyler, Matthew Peveler, and Barbara Cutler. "A Flexible Late Day Policy Reduces Stress and Improves Learning". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, Mar. 8, 2017, p. 718. ISBN: 978-1-4503-4698-6. DOI: `10.1145/3017680.3022439`. URL: `https://doi.org/10.1145/3017680.3022439` (visited on 03/17/2025).

[54] Elise Vambenepe et al. "Impact of Retake Policy on Student Performance in a CS0 Course with Mastery Learning". In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, Feb. 18, 2025, pp. 1643–1644. ISBN: 979-8-4007-

0532-8. DOI: 10.1145/3641555.3705199. URL: https://doi.org/10.1145/3641555.3705199 (visited on 05/12/2025).

Appendix A

GradeSuite Handoff Plan

The *GradeSuite* project is an ongoing project that has been in development for the past two years, and will continue to be actively developed and maintained at UC Berkeley for the foreseeable future. Notably, multiple microapps are currently in development by their respective teams, which will be added to the *GradeSuite* project in the coming months and years. The below sections outline the process by which a new Organization Lead can take over the *GradeSuite* project and its respective microapps.

When selecting a new organization lead, it is recommended that they have the following qualifications:

- Prior project management experience
- Cloud service architecture
- Experience with CI/CD pipelines and workflow automation

- Experience with secure software development practices
- The following languages:
  - JavaScript
  - Python
- The following technologies and frameworks:
  - Git and GitHub
  - Docker
  - Google Cloud Platform
  - Linear, Jira, and/or a similar issue tracking system
  - Linux and/or Unix-like systems
  - Node.js (preferred)
  - React (preferred)
- Web servers (e.g. nginx) (preferred)
- Systemd services (preferred)

A.1 GitHub

The GitHub organization is currently owned only by the organization lead. The new *GradeSuite* organization lead will require ownership access to the organization in order for

them to manage it as they see fit. The new organization lead will need to manage multiple elements of the GitHub organization, including:

- Organization teams
- Organization members and their respective team memberships
- Repository access and permissions
- Authorized applications and integrations including but not limited to:
- Branch protection rules which enforce the pull request etiquette outlined in section 3.3
  - CodeCov which is used for test coverage in certain microapps
  - GitGuardian which is globally configured for automated security scanning and general-purpose linting
  - Linear which is used for automated issue tracking and ticket status updates
  - Slack which is used for organization communication and automated pull-request review notifications

They will also be responsible for updating the billing and contact information for the organization (note: the organization is currently configured as a free organization, so no payment information is required, but the billing email should be updated regardless). They will also be responsible for the repository named '.github' which contains organization meta-data and shared resources such as the PR template and landing-page 'README.md' file.

A.2 Google Cloud Platform

The primary contact for access within the Google Cloud account is the organization lead. They have ownership access to the 'eecs-gradeview' namespace within the 'berkeley.edu' Google Cloud organization. This namespace contains all of the *GradeSuite* related resources.

To grant access to the cloud console for new developers, the new organization lead will need to add the new user as a principal through iam with the role 'AFA Cloud Developer.' General access requests that apply to all developers should be added to the 'AFA Cloud Developer' role directly, and more specific, protected, access requests should be handled on a case-by-case basis and applied directly to the user's principal to ensure least-privilege access.

The organization lead will also be the default on-call user for *GradeSuite* services — it is recommended that monitors configured for other microapps have proper escalation procedures to ensure that the organization lead is notified in a timely manner, and that allow them to delegate on-call duties to other team members as needed. Moreover, it is the organization lead's responsibility to ensure that the alerting policies are properly integrated with other organization systems such as Slack, to inform the overall team and relevant stakeholders of any issues or outages that may occur. These monitors and policies should be configured through Google Cloud's monitoring and alerting tools.

A.3 Slack

The Slack workspace is configured with a variety of different integrations and bots which are used to facilitate organization communication and automation. The new organization lead will need to manage these integrations and bots, including but not limited to:

- GitHub
- Google Cloud Platform (Alerting and Monitoring)
- Linear

It is likewise the duty of the organization lead to set up channels for each microapp and validate the above integrations for respective git repositories. Moreover, they are expected to actively monitor and moderate the slack workspace regularly including all microapp team channels.

A.4 Linear

The Linear project board is actively owned and managed by the organization lead. For proper handoff, they will need admin access to the Linear project board which can be provisioned only by the existing organization lead.

Section 3.3 outlines the use of Linear for issue tracking and ticket status updates. The new organization lead will need to manage the Linear project board and its respective workflows.

Specifically, they will need to configure the teams and projects for each microapp. They are also responsible for providing new developers access to the Linear project board and assigning them to the appropriate teams and projects. Likewise, it is the organization lead's responsibility to ensure the Linear board actively reflects the overall organization's goals and objectives as laid out in the weekly meetings with the entire research group. They should monitor the board regularly to confirm tickets are being updated and closed in a timely manner — these updates can also be reflected in the organization's weekly team meetings.

Linear is also configured to integrate directly with GitHub and slack as mentioned in section A.1 and section A.3 respectively. These integrations are likewise set up in Linear directly, so the new organization lead will need to maintain and update these integrations as needed.

A.5 Domain Name and DNS Access

The DNS is not currently owned or managed by the *GradeSuite* team at all, but rather by the University of California, Berkeley Information Technology Services team. As such, any changes or updates to the DNS will need to be made through the respective IT personnel whom can be reached at `inst@eecs.berkeley.edu`. It should be configured to point to the Google Cloud gateway IP address and should properly resolve to the *GradeSuite* website (hosting the *GradeView* frontend on web).

A.6 Microapp Handoff

The *GradeSuite* organization lead must collaborate with the research group lead to select effective team leads for each microapp. These leads will not only be responsible for the day-to-day development and maintenance of the microapp, but will also be responsible for onboarding and training new developers on their team. Moreover, the organization lead must also collaborate with the research team lead at the beginning of their term to explicitly lay out the organization's goals and key performance indicators each semester which will inform the workload distribution and therefore the human resources needed for each microapp.

Moreover, the organization lead is also responsible for the overall performance of the *GradeSuite* project, meaning they should have a deep understanding of the project's infrastructure and capabilities, specifically the architecture referenced in Figure 4.2. They will be responsible for ensuring that the microapps are properly integrated with each other and that the overall architecture is up-to-date and properly configured.

Microapp Lead Responsibilities

Each microapp lead also assumes a variety of responsibilities. Primarily, their goal is to evaluate project deliverables and coordinate with their team to ensure that each deliverable is of high quality and meets the needs of the organization. In order to most efficiently accomplish this, they should use the Linear project board to break down large asks into epics and stories which can be assigned to individual team members. These tickets are the

main point-of-reference for evaluating team velocity.

Microapp leads are also expected to attend the weekly meetings with research team to ideate on future development projects as well as weekly meetings with the organization lead to discuss the technical direction of the project and outstanding deliverables. They should also schedule at least one meeting per week with their respective teams outside of these larger meetings to expedite the development process and address any blockers that may arise.

If delays are anticipated, the microapp lead should communicate with the organization lead as soon as possible to discuss potential solutions. If the issue is not resolved within a reasonable amount of time, the microapp lead should work with the organization lead to escalate the issue to the research team lead and any other relevant stakeholders.

In the event that a microapp lead steps down or a new lead is appointed, the existing microapp lead is expected to work with the new lead to ensure a smooth transition. This may include training the new lead on the microapp's infrastructure, documentation, and any other relevant information.

Appendix B

Survey Questions

Throughout the semester, we collected feedback from students in two ways: an end-of-semester survey that course staff directly distributed to students, and a general feedback form that students could use during the semester to provide feedback and report bugs as the semester progressed.

B.1 End-of-semester Survey

The end-of-semester survey was primarily comprised of two sections: *GradeView* and *Concept Map*. This is because these were the two services that students directly interacted with. The related survey questions are included below.

GradeView / GradeSuite

Please complete this section based on your experience with the *GradeView* platform.

1. How often do you use *GradeView*?

   ○ Daily

   ○ Weekly

   ○ Monthly

   ○ Never

   ○ N/A (didn't use *GradeView*)

2. What do you primarily use *GradeView* for?

   ○ Viewing grades

   ○ Seeing how I'm progressing relative to the class

   ○ Getting general course information

   ○ Viewing the concept map

   ○ N/A (didn't use *GradeView*)

   ○ Other (please specify)

3. How many questions/assignments have you retaken based on scores shown in *GradeView*? (If you did not use *GradeView*, please enter "NA")

4. How many questions/assignments did you anticipate retaking, but chose not to based on scores shown in *GradeView*? (If you did not use *GradeView*, please enter "NA")

Please rank the following questions with how you feel best describes your experience with *GradeView*.

5. I was well informed about how I can use *GradeView* in the context of the class.

   ○ Strongly disagree

   ○ Disagree

   ○ Neutral

   ○ Agree

   ○ Strongly agree

6. *GradeView* helps me track my performance in this class.

   ○ Strongly disagree

   ○ Disagree

   ○ Neutral

   ○ Agree

   ○ Strongly agree

   ○ N/A (didn't use *GradeView*)

7. *GradeView* helps me improve my performance in this class.

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree
- ○ N/A (didn't use *GradeView*)

Other

8. Have you noticed any incorrectly reported grades on the *GradeView* platform?

   - ○ Yes
   - ○ No
   - ○ N/A (didn't use *GradeView*)

9. Please use this space to give us any additional feedback about the *GradeView* system.

Concept Map

Please answer the following questions about *Concept Map* in specific.

10. How often do you use *Concept Map*?

    - ○ Daily

○ Weekly

○ Monthly

○ Rarely/Never

11. Did *Concept Map* influence your decision of whether or not to retake an assignment? Please select N/A if you do not use *Concept Map*.

○ Yes

○ No

○ N/A (didn't use *Concept Map*)

12. I understood how to interpret the data presented in *Concept Map*.

○ Strongly disagree

○ Disagree

○ Neutral

○ Agree

○ Strongly agree

○ N/A (didn't use *Concept Map*)

13. *Concept Map* helps me track my progress in the class.

○ Strongly disagree

- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree
- ○ N/A (didn't use *Concept Map*)

14. *Concept Map* helps me pace myself through course content.

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree
- ○ N/A (didn't use *Concept Map*)

15. *Concept Map* helps me improve my performance in the class.

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree

○ N/A (didn't use *Concept Map*)

16. Have you noticed any incorrect data on *Concept Map*?

    ○ Yes

    ○ No

    ○ N/A (didn't use *Concept Map*)

17. Please use this space to give us any additional feedback about *Concept Map*.

B.2 General Feedback Survey

We also provided students with a general feedback form when announcing the launch of *GradeSuite* in their course offering. They could use this form to make bug reports, feature requests, or other general feedback; each topic consisted of its own subset of questions:

Bug Report

Sorry you're running into issues. Let us know whats going on so we can fix it ASAP!

1. What page is the bug you are running into on?

2. What is the expected behavior? (what do you expect should be happening)?

3. What is the actual behavior?

4. Briefly describe any further details

5. Please add any images of the issue here

6. Are you willing to go through some brief troubleshooting steps to help us better diagnose and resolve the issue?

   ○ Yes

   ○ No

If the students answered "Yes" to the last question, we would then walk them through how to download their network logs as a `.har` file, and upload them directly to us. These were useful for diagnosing issues with early versions of *GradeSuite*, before we had comprehensive logging in place.

Feature Request

Thanks for spending the time to request a feature! Let us know what we should add :)

1. Describe your feature

General Feedback / Other

Thanks for giving us some feedback about your experience with the app!

1. Let us know your thoughts below!

Final Questions

The last two questions on the form, regardless of their feedback type, were a general questions:

1. Please rate your experience with *GradeView* so far

    ○ 1 star

    ○ 2 stars

    ○ 3 stars

    ○ 4 stars

    ○ 5 stars

2. Would you like us to reach out directly to you about your submission?

    ○ Yes

    ○ No

Appendix C

Source Code

The source code for this project and each of its respective microapps can be found in the GitHub organization at `https://github.com/AFA-Tooling` [2]. The below listings are included for convenience and are referenced elsewhere in this document.

```
## Linear Ticket

<!-- Replace the field marked <ticket-id> with your ticket id -->
[Linear Ticket](https://linear.app/afa-tooling/issue/<ticket-id>)

### Description

<!-- Briefly describe the feature being introduced. -->

### Type of Change

<!-- What types of changes does your code introduce? Put an `x` in all the boxes that apply. -->

- [ ] Bug fix (non-breaking change which fixes an issue)
- [ ] New feature (non-breaking change which adds functionality)
- [ ] Refactoring (non-breaking change)
```

```
17 - [ ] Breaking change (fix or feature that would change
     ↪ existing functionality)
18
19 ### Changes
20
21 <!-- List the major changes made in this pull request. -->
22
23 ### Testing
24
25 <!-- Describe how the feature has been tested, including both
     ↪ automated and manual testing strategies. -->
26
27 ### Checklist
28
29 - [ ] My branch name matches the format: `<ticket-id>/<brief-
     ↪ description-of-change>`
30 - [ ] My PR name matches the format: `[<ticket-id>] <brief-
     ↪ description-of-change>`
31 - [ ] I have added doc-comments to all new functions ([JSDoc](
     ↪ https://jsdoc.app/) for JS and [Docstrings](https://peps.
```

```
   ↪ python.org/pep-0257/) for Python)
- [ ] I have reviewed all of my code

### Screenshots/Video

<!-- Include screenshots or video demonstrating the new feature
   ↪ , if applicable. -->

### Additional Notes

<!-- Any additional information or context relevant to this PR.
   ↪  -->
```

Listing C.1: Pull Request Template.

```
{
    "redis": {
        "username": "default",
        "host": "redis",
        "port": 6379
    },
    "spreadsheet": {
        "id": "13qkZrlXtCWIqPI-0N86s-ZLtw9DTvYVzPI5BnJZ5gaM",
        "scopes": [
            "https://www.googleapis.com/auth/spreadsheets.
    ↪ readonly"
        ],
        "pages": {
            "gradepage": {
                "pagename": "HAID",
                "assignmentMetaRow": 2,
                "startrow": 4,
                "startcol": "C"
            },
            "binpage": {
```

```
                "pagename": "Constants",
                "startcell": "A51",
                "endcell": "B61"
            }
        }
    },
    "googleconfig": {
        "oauth": {
            "clientid": "435032403387-5
↪ sph719eh205fc6ks0taft7ojvgipdji.apps.googleusercontent.
↪ com"
        }
    },
    "admins": [
        "connorbernard@berkeley.edu",
        "ddgarcia@berkeley.edu"
    ]
}
```

Listing C.2: *GradeView* API Configuration File.

```
name: CS10
term: Fall 2024
orientation: left to right
start date: 2024 08 26
styles:
    name: root, shape: ellipse, style: filled, fillcolor: #3
    ↪ A73A5
    name: blue1, shape: ellipse, style: filled, fillcolor: #74
    ↪ B3CE
    name: blue2, shape: ellipse, style: filled, fillcolor: #87
    ↪ CEEB
    name: default, shape: ellipse, style: filled, fillcolor: #
    ↪ E0E0E0
class levels:
    Not Taught: #dddddd
    Taught: #8fbc8f
student levels:
    First Steps: #dddddd
    Needs Practice: #a3d7fc
    In Progress: #59b0f9
```

```
    Almost There: #3981c1
    Mastered: #20476a
nodes:
    Quest [blue2, Week1]
        Abstraction [default, Week1]
        Number Representation [default, Week1]
        Iteration [default, Week1]
        Domain and Range [default, Week1]
        Booleans [default, Week1]
        Functions [default, Week1]
        HOFs I [default, Week1]
    Midterm [blue2, Week5]
         Algorithms [default, Week5]
         Computers and Education [default, Week5]
         Testing + 2048 + Mutable/Immutable [default, Week5]
         Saving the World with Computing [default, Week5]
         Debugging [default, Week5]
         Scope [default, Week5]
         Iteration and Randomness [default, Week5]
         Recursion Tracing [default, Week5]
```

```
37          Algorithmic Complexity [default, Week5]
38          HOFs II [default, Week5]
39          Fractal [default, Week5]
40      Projects [blue2, Week1]
41          Project 1: Wordle-lite [default, Week1]
42          Project 2: Spelling Bee [default, Week1]
43          Project 3: 2048 [default, Week1]
44  end
```

Listing C.3: CS10 Concept Map Syntax.