A Platform for Data Analysis and Tutoring For Introductory Programming

A Thesis Submitted

in partial fulfilment of the requirements
for the degree of

Master of Technology

by

Rajdeep Das

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

June, 2015

CERTIFICATE

It is certified that the work contained in the thesis titled **"A Platform for Data Analysis and Tutoring For Introductory Programming"**, by **Rajdeep Das**, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Amey Karkare 8/5/2015

Prof Amey Karkare
Department of Computer Science & Engineering
IIT Kanpur

Sumit

Dr Sumit Gulwani
Microsoft Research
Redmond

June, 2015

ABSTRACT

Computer aided education[1] is the emerging technology to make education available to masses all over the world. MOOC[2] platforms have solved the problem of managing large courses, improving community interactions and making them available to people covering a large geographic area. However, they do not help the students much when solving problems offered by the course. Problem solving is an integral part of learning in courses, and MOOCs have not added any value to the existing methods present today. In this thesis, I will attempt to tackle such issues in introductory programming courses. The issues that arise here, with respect to problem solving include lack of early and uniform feedback, necessity to learn a complex programming environment and lack of information about students' approach to solving problems. I will present a tutoring system platform which may be used to conduct introductory programming courses. The system provides a cloud based web application which can be used to learn programming and provides instant feedback to students while solving programming problems. The system also provides a view of the students' approach to solving programming problems. It avoids the requirements for students, of having to learn a programming environment such as Linux and its tools, and enables the students to focus on solving only the programming problems. The system also collects valuable data regarding how the students solve the problems, the errors they commit and a lot of other information valuable for analysis. The various interfaces in this system provide a lot of analytics for this collected data. Such data is then used in developing intelligent tools for giving feedback to students, some of which are used in this system itself. This system thus serves as a platform for tutoring as well as data collection for researchers.

[1] Can Cemal Cingi, "Computer Aided Education," *Procedia - Social and Behavioral Sciences* 103 (November 2013): 220–29, doi:10.1016/j.sbspro.2013.10.329.
[2] "Massive Open Online Course," *Wikipedia*, n.d., http://en.wikipedia.org/wiki/Massive_open_online_course.

Dedicated to my parents

ACKNOWLEDGEMENT

I would like to express my sincere gratitude towards my thesis supervisor Prof. Amey Karkare for providing me with all the support and encouragement needed to accomplish this thesis. Without his help in providing the required infrastructure and authorizations, it would not have been possible to carry out the necessary experiments for this thesis work. I am also overly thankful to him for supporting the deployment of this system at IIT Kanpur for the ESC101 course which is taught to 400 odd students from various disciplines. He had given me the wonderful opportunity of being able to architect and deploy a full blown system on powerful systems, to be used by live users.

I would like to thank Dr Sumit Gulwani, who is my co-supervisor for this thesis, for his extremely valuable guidance into solving the problems encountered by me while doing this thesis. It would not have been possible for me to complete this thesis in time and come up with meaningful results, if it were not for his guidance. He has been the backbone of driving this project forward through our weekly meetings, and giving valuable feedback on every minute aspect of this system.

I would finally like to thank a few other people whose efforts were no less valuable in developing this thesis project. I would like to start with Umair Z Ahmed for providing a lot of help in conducting the ESC101 course and also providing valuable insights in conducting the course using this system. I would like to thank Naman Bansal for his contributions to automatic test case generation using KLEE, which has helped generate a large number of test cases for the problems used in the course. I would also like to thank all the undergraduates, who had taken up internships for this project in summer, for their valuable contributions to problem generation for the ESC101 course.

Table of Contents

LIST OF FIGURES

INTRODUCTION

Computer aided education is the emerging technology to enhance the process of learning and make education available to masses all over the world. Several software systems have been built to enhance the process of learning across various domains. MOOC[3] platforms are a very prominent example which has been primarily built to enhance reachability to students distributed across the world. Examples of MOOCs include Coursera[4], Edx[5], Khan Academy[6], Udacity[7], etc. These online platforms host a large number of courses, conducted by instructors from prominent institutes all over the world. Such systems have also succeeded in providing open courses to the masses and improving community interaction whilst learning. However problem solving forms an integral part of learning in courses, and the problem of providing early and uniform feedback to students when solving problems in courses remains persistent even now. Moreover these MOOCs are quite generic in nature and do not address the problems of conducting individual courses, such as programming. Some of these problems include the necessity for students to learn a programming environment such as Linux and its build tools. Most students who learn programming do not need to know of these environments as there is a lot of Integrated Development Environments (IDE)[8] available today, which abstract these issues from the programmers. Further, when solving a programming problem, sometimes it is quite necessary to determine the approach of the student while solving a programming problem. This helps to judge the solution presented by the student as well as provide feedback to the student when she commits errors.

In this thesis, I will try to solve the problems mentioned above from an architectural point of view, by presenting a tutoring platform which may be primarily used to conduct introductory programming courses. The platform can be used to deploy a system which can be used to teach programming to students by providing a versatile programming environment and real-time feedback systems. The system provides a cloud based editor interface, which can be accessed from any standard browser. This abstracts the issues of having to learn a complex programming environment such as Linux, in order to learn programming. The system enables easy integration of feedback tools so that early and uniform feedback can be provided to the students. The system also records code progression of student programs, which help to decipher the approach of the student when solving programming problems. From a broad perspective, the platform is intended to be used by students, instructors, developers and researchers combined to achieve their respective goals. Students would use this system to learn introductory programming and benefit most of the features available in MOOCs. Instructors would use this system to conduct introductory programming courses. Developers may use this platform as a framework to create tutoring systems in programming. Finally, researchers may use this system

---

[3] Ibid.
[4] “Coursera.org,” n.d., https://www.coursera.org/.
[5] “Edx,” n.d., https://www.edx.org/.
[6] “Khan Academy,” n.d., https://www.khanacademy.org/.
[7] “Udacity,” n.d., https://www.udacity.com/.
[8] “Integrated Development Environment,” *Wikipedia*, n.d., http://en.wikipedia.org/wiki/Integrated_development_environment.

as a tool for collecting valuable data to understand patterns in the student learning process. This versatile system is tested in production for the introductory programming course at IIT Kanpur, and meets the goals of every user mentioned above.

This document essentially describes the software architecture of “A Platform for Data Analysis and Tutoring for Introductory Programming”. It is intended for software developers and implementers of the platform along with anyone who needs to understand the working of this software platform. The document covers the description of the platform as well as critical reasoning, in order to help the audience realize the pros and cons of deploying this platform. It also provides developers with an idea of how to extend or modify this platform. Researchers may also be able to understand how to use this platform for data analysis and research in the area of Intelligent Tutoring Systems. This document may be used as an architecture reference manual for developers as well as an architecture overview for a wide range of users. It is presented in a style designed by HP[9].

[9] Michael A. Ogush, Derek Coleman, Dorothea Beringer, “A Template for Documenting Software and Firmware Architectures,” March 15, 2000, http://www.cs.helsinki.fi/group/os3/HP_arch_template_vers13_withexamples.pdf.

RELATED WORK

A large number of premier institutions have been conducting MOOCs since quite some time which had led to the development of popular online educational systems. Coursera emerged in 2012 which now offers more than 1000 courses across 117 institutions according to Wikipedia[10]. Other notable instances include Khan Academy which was founded by educator Salman Khan in 2006, EdX by MIT and Harvard founded in 2012 and Udacity founded by Sebastian Thrun, David Stavens, and Mike Sokolsky. In India, NPTEL[11] which is funded by the Ministry of Human Resource Development (MHRD) is managed by seven IITs and IISc Bangalore. All these examples offer MOOCs, which aim at teaching a large population of students for free.

Intelligent tutoring systems have also been existent for some time now. The automata tutor[12] developed initially by Arjun Radhakrishna, and Damien Zufferey helps students learn basic concepts of automata theory. It is based off the papers at IJCAI '13[13] and TOCHI '15[14]. This system uses tools which generates feedback for DFA assignments for automata courses. It is also capable of automatic grading of such assignments. Even in the area of programming, there have been advancements in development of feedback tools. AutoProf[15] is an example, which generates feedback for programming assignments in introductory programming courses. It has been integrated into the EdX course "Introduction to Computer Science and Programming (6.00x)". A logic tutor[16] had also been built for the generation of natural deduction problems using first order logic. A large number of these tools have been built using techniques from program synthesis. Such techniques have been used in problem generation[17], solution generation[18] and automated grading[19] of assignment problems from a large set of disciplines.

---

[10] "Coursera," *Wikipedia*, n.d., http://en.wikipedia.org/wiki/Coursera.
[11] "NPTEL," n.d., http://nptel.ac.in/.
[12] "Automata Tutor," n.d., http://www.automatatutor.com/.
[13] Rajeev Alur et al., "Automated Grading of DFA Constructions," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13 (Beijing, China: AAAI Press, 2013), 1976–82, http://dl.acm.org/citation.cfm?id=2540128.2540412.
[14] Loris D'antoni et al., "How Can Automatic Feedback Help Students Construct Automata?," *ACM Transactions on Computer-Human Interaction* 22, no. 2 (March 10, 2015): 1–24, doi:10.1145/2723163.
[15] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama, "Automated Feedback Generation for Introductory Programming Assignments," *ACM SIGPLAN Notices* 48, no. 6 (June 23, 2013): 15, doi:10.1145/2499370.2462195.
[16] Umair Z. Ahmed, Sumit Gulwani, and Amey Karkare, "Automatically Generating Problems and Solutions for Natural Deduction," in *IJCAI 2013*, 2013.
[17] Oleksandr Polozov, Sumit Gulwani, and Sriram Rajamani, *Structure and Term Prediction for Mathematical Text* (Microsoft Research, 2012).
[18] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari, "Synthesizing Geometry Constructions," in *PLDI*, 2011, 50–61.
[19] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama, *Automated Semantic Grading of Programs*, 2012.

SYSTEM PURPOSE

The platform provides web application interfaces which is targeted for both students and instructors. Students are provided with an editor with integrated tools for compilation, execution, evaluation and feedback generation. They are also provided basic course related statistics such as grades. Instructors are provided with interfaces to manage problems, user accounts, and course events such as labs, exams and quizzes. There are other features that the interfaces offer, which will be elaborated later in this document. The tutoring system is designed to support a class of students having strength of around 500. It is designed for high availability and moderate performance.

Context

Current MOOC systems offer basic course management through online portals. They still require a lot of tedious human effort to provide high quality education through it. In specific cases of programming courses, this effort can be substantially decreased through the introduction of automated and intelligent tools for tasks such as student thought process analysis, problem generation and feedback generation.

This tutoring system solves the problem of teaching introductory programming to a large population of students using automated feedback generation, data analytics and course management. It enables instructors to provide high quality feedback to students in real time, while solving programming problems, using some intelligent feedback generation tool. It also enables management of programming courses. A key feature of this system is that it provides interfaces for viewing the code progression of student programs. This helps in analysing the thought process of the students while programming, and provides an excellent base for research and analytics. The system adds value to existing MOOCs by providing real-time feedback generation through the versatile code editor and also by providing a platform for data analysis.

This software platform will be used by instructors to teach students introductory programming using an intelligent tutoring environment. It may also be used by researchers to analyse data collected from the students, in order to come up with new techniques for intelligent tutoring.

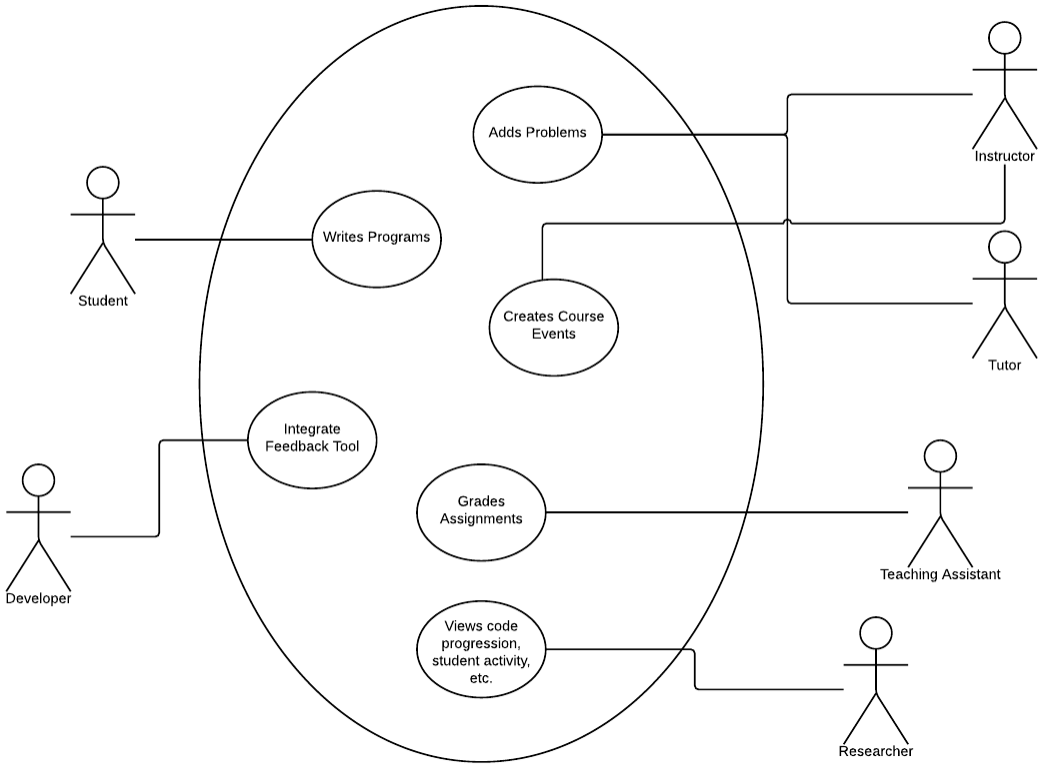

Figure 1: High level use-case diagram of Tutoring System

System Interface

Interface	Student home
Use Cases	**View grades and course statistics:** This interface allows student users to view their grades for previous course assignments. It also enables them to view their own statistics for the course, such as the number of submitted solutions. **View on-going events:** This interface displays any on-going course event along with a summary of the problems that have been assigned to the student for the respective course event. A student can directly go to the environment from where he can start coding for any of the problems listed in the summary.

Interface	Editor for course events

Use Cases	**View problem:** This interface displays a programming problem that has been assigned to the respective student for the current on-going course event. The student is able to view this problem from one of the tabs located to the left of the user interface. **Write programs:** This service allows student users to write source code for the programming problem that is defined within this interface. Code can be written in an editor that is displayed within this interface. The editor support basic features such as syntax highlighting, code-folding, automatic indentation and displaying line numbers. Students are also able to save their programs manually. **Compile programs:** This service allows student users to compile the programs that they have written in the editor. They are also able to view the compiler messages in a console window within this interface, along with any feedback that are provided by the feedback tools integrated into this system. **Execute programs:** This service allows users to execute compiled programs that they have written in the editor. They can supply input data into a corresponding window located within this interface. The output from the program is displayed in another similar window located adjacently. This interface also displays any error messages that may have occurred while executing the program. **Evaluate programs:** This service allows users to evaluate their program against a set of test cases supplied along with the problem statement. They are able to view the number of passed test cases along with a table showing which have passed and which have failed, if the respective test cases are visible. The interface also displays any feedback generated by any of the relevant feedback tools integrated into this system. **Submit programs:** This service allows student users to submit their programs as solutions for the current programming problem.

Interface	Scratchpad

Use Cases	**Create/Delete files/folders:** This service allows users to create and delete virtual source files, and folders to organize them. **Write Programs:** This service allows users to write arbitrary programs for practice or testing purposes. An editor is provided within this interface, which is similar to the one present in the editor for course events. These programs are saved in the virtual source files created above. **Compile:** This service allows users to compile their source code, written in the virtual source files, and view the messages generated by the compiler. **Execute:** This service allows users to execute their compiled source code on custom input, and view the results in a corresponding output window. Users are also able to view any errors which occur while executing the program.

Interface	Codebook
Use Cases	**View attempted problems list:** This service allows users to view the list of practice problems that they have attempted, and also the list of problems they had been assigned during the course events. Students can view these problems and their own solutions to them. **View submitted solution:** This service allows students to view the code submitted as solution for the problems listed within this interface. For practice problems, this interface displays the last saved solution code. It also displays the grading status of the problem, if the problem is an assignment problem.

Interface	Practice arena
Use Cases	**View practice problems:** This service allows student users to view the problems that have been marked for practice. Students can directly navigate to the “Editor for course events” interface from here, for the corresponding problem.

Interface	Pager send message
Use Cases	**Start support thread:** This service allows student users to send a message to all concerned teaching assistants, tutors and instructors. Messages are expected to contain queries regarding difficulties in using the system and in solving assignment problems. References to source code versions are automatically included in the messages, if they are sent from the editor interface for course events.

Interface	Pager viewer
Use Cases	**View and respond to replies:** This service allows student users to view the replies to the messages sent by them through the “Pager send message” interface. They can also respond to such replies, thus continuing a message thread.

Interface	Admin home
Use Cases	**Navigate to management and analytics interfaces:** This interface is used by administrators to navigate to the management and data analytics interfaces. It provides an overview of what features are available to the user.

Interface	Problem management
Use Cases	**Create problems:** This service is used to create a new blank problem instance by specifying an identifier for the problem, along with a category that the problem belongs to. **View/Edit problems:** This service is used to view and edit previously created problems. Items that can be edited for a problem are its statement, solution code and solution template. All edits are reflected instantly to anyone viewing the respective problem. **Add/Delete test cases:** This service allows instructors, tutors and teaching assistants to add test cases for the problem, upon which student solutions can be evaluated. Test

	cases can be added individually, or in batches. Test cases can also be deleted using this interface.

Interface	Problem upload
Use Cases	**Upload problems in batch:** This service allows problems to be uploaded to the system in batch. The problems are required to be in a specific format, in order to be uploaded.

Interface	Account management
Use Cases	**Add accounts:** This service allows administrators to add user accounts to the system. User accounts may be either student or administrator accounts. **Edit/Delete accounts:** This service allows administrators to edit profile and accounting information of existing users, including authentication methods. It also allows deleting of user accounts from the system. For administrator accounts, this interface also facilitates changing of administrator roles such as instructor, tutor and teaching assistant.

Interface	Event management
Use Cases	**View course event calendar:** This service allows viewing of course events and their corresponding schedules. The interface includes a calendar, which helps instructors to plan course events. **Create course events:** This service allows the creation of new course events in the system, such as labs, exams and quizzes. **Create event schedules:** This service allows instructors to create schedules for the course events which had been created previously. **Assign problems to students:** This service allows instructors to assign problems to students for course events that had been created previously. This assignment is done based on some algorithm, which is interactively specified by the instructor.

Interface	Event dashboard
Use Cases	**View student performance rankings:** This service allows instructors to view rankings of students according to some score computed using some algorithm. **View student performance distributions:** This service allows instructors to view the distribution of the scores mentioned above.

Interface	Submissions viewer
Use Cases	**View student submissions:** This service enables viewing of student submissions to problems for course events. The submissions can be filtered based on a set of categories.

Interface	Code history viewer
Use Cases	**View code history:** This service allows viewing of source code history for a specific course assignment. It is used to view the stimuli on which code versions were saved along with the code that was saved. **View submission code:** This service allows viewing of the code version that was submitted by the student, while solving this assignment. **Evaluate submitted code:** This service enables evaluation of the submitted source code on the test cases that are available for the respective problem. Evaluation results for each test case are displayed in a table. **Grade submissions:** This service enables teaching assistants and tutors to grade the assignment being viewed.

Interface	Admin editor
Use Cases	**Compile program:** This service allows admin users to compile programs and view syntactic error messages. **Execute program:** This service allows admin users to execute programs on custom input. **Evaluate program:** This service allows admin users to evaluate programs that correspond to a student submission for a

	course event. Admins are able to modify the code and then evaluate on the modified code. **Update solution code:** This service allows admin users to update the ground truth solution code to a particular problem.

Interface	Assignment analytics
Use Cases	**View code size variation:** This service allows users to view the variations in code size of a student submission over time. **View code save progression:** This service allows users to view the progression of code saves for a student submission, over time. **View syntactic analytics:** This service allows users to view the sort of compilation errors performed by a student while solving that particular programming assignment. It also shows instances of the errors and where they were triggered. **View compilation error timeline:** This service allows users to view the compilation errors committed by the students in solving the problem, as well as the time taken by them to fix the individual compilation errors. **View execution sequence:** This service allows users to view the sequence of executions and evaluations made by a student while solving a programming assignment, over time. It also displays the results of those executions and evaluations.

Interface	Dataviz
Use Cases	**View visualizations and clusters:** This interface allows users to view the data collected by the system after some processing. The format in which the data is displayed depends on the developers of the visualizations. For example, it can be used to view the compiler messages corresponding to the various student programs.

Interface	Control panel
Use Cases	**Modify compiler options:** This service allows admins to modify the flags used by the

	compiler to compile programs. **Modify execution sandbox settings:** This service allows admins to change the quotas for execution of programs. Quotas include time and memory. **Enable/Disable plugins:** This service allows admins to enable or disable plugins used in the system. **Modify delays:** This service allows admins to modify the time delays for compilation, execution and evaluation of programs. **Enable/Disable logging:** This service allows admins to enable or disable logging of compilations, executions and evaluation attempts by students, while solving programming problems.

Interface	Tasks panel
Use Cases	**View personal pending/complete tasks:** This service allows admin users to view the pending or complete tasks, which have been assigned to them. **View overall pending/complete tasks:** This service allows admin users to view the status of tasks that have been assigned to all admin users, grouped by the admin users and the course events for which tasks have been assigned.

Interface	Admin pager
Use Cases	**View messages:** This service allows admin users to view the message threads created by students, as well as all replies to such threads. **Reply to messages:** This service allows admin users to reply to the messages that are viewable in this interface. Users can also navigate to the code instance which the student was working on (if any), when he sent the message.

Interface	Admin account settings
Use Cases	**Update name:** This service allows admin users to update their name on the system.

	Update password: This service allows admin users to update their password on the system.

Interface	Cache interface
Use Cases	**Store/Retrieve session/cache data:** This interface allows the web and engine applications to store and retrieve data corresponding to session information and database rows for caching.

Interface	Database proxy
Use Cases	**Communicate with database nodes:** This interface allows the web and engine application services to access database nodes for executing queries on them.

Interface	Database node interface
Use Cases	**Communication with database server:** This interface allows the database proxy and optionally the web/engine applications to communicate with the database server running on that node.

Interface	Web application interface
Use Cases	**Retrieve user interfaces:** This interface is used by browser clients to retrieve user interfaces provided by the web application services. **Send RPC requests:** This interface is used by the web application subsystem to send send RPC requests to the web application service.

Interface	Engine endpoint
Use Cases	**Compile/Execute/Evaluate:** This service is used by the web application to send requests for compilation, execution or evaluation to the engine. **Run tools:** This service is used by the web

	application to run various tools hosted by the engine.

Interface	HTTP proxy
Use Cases	**Access web services:** This interface is used by the clients to access web services offered by the tutoring system.

Interface	Static proxy
Use Cases	**Access static content:** This interface is used by the web application to access static content such as stylesheets, scripts and images.

Interface	Consul interface
Use Cases	**View health status of nodes:** This service allows users to view the health status of the various nodes deployed in the system.

Non-Functional Requirements

Qualities

- **Portability:** The system is designed to run on almost any flavour of Linux. The nodes can be packed into archives and relocated to different physical machines. Only the environment which is required to run the nodes must be set up on the physical machines. Docker is the environment which is used for deploying nodes, and can easily be set up on any standard Linux distribution.
- **Extensibility:** The system and its functionalities can be extended by adding new plugins and tools to it. Adding plugins for specific purposes such as feedback generation requires a few lines of configuration and code. Other tools can be integrated by adding modules, which are quite simple to write, and will require less than a day to implement for most scenarios. Also new features can be added to the system without any downtime.
- **Modifiability:** The existing tools and modules in the system can be modified quite easily by changing a few lines of code. Few tools such as compilers can be changed by simply changing the compilation configurations. Features can also be enabled and disabled via the admin interface using simple clicks.
- **Scalability:** New nodes can be added and removed to and from the system without any downtime. New physical machines can also be added to the system, without affecting

the uptime. The system has the capability to auto scale nodes within a physical machine, in the case when the machine is used for multiple purposes.

- **Usability:** The system is meant to be used by novice computer users to expert programmers. The user interfaces are modular and intuitive.
- **Durability:** The data nodes on the system are replicated, thus making it immune to data loss.
- **Fault-Tolerance:** All nodes on the system are replicated, thus making the system highly available. Failure of nodes does not result in downtime due to the presence of redundant replicas. The proxies also operate in high availability master-slave mode, thus adding a new layer of tolerance at the physical level.

Constraints

- The system has to run on a Linux environment. However any bare minimal Linux distribution which supports Docker will suffice.

Principles

- The entire system is highly modularized. This enables the system to be easily modifiable. New features can be added by introducing new modules and registering them on the system. This principle supports extensibility and modifiability.
- The nodes of the system are deployed using Docker. Each node is a container spawned from a previously created image. Since Docker can be archived and ported from machine to machine, so can the system. This principle supports portability.
- The systems services are separated into different nodes. Nodes hosting specific services are clustered. Additional nodes can be added to the system to increase performance, and nodes can also be removed from the system to free up resources. This principle supports scalability.
- The database nodes are clustered for replication. Each node is a replica of the others in the system. All writes to the nodes are synchronous, thus ensuring that data is replicated to all of the nodes, every time a write occurs on one of the nodes. Hence, data is not lost in the event that any one of the nodes crash. This principle supports durability.
- The physical machines have proxies that listen on a single floating IP with a master-slave configuration. Thus, in the event that one of the physical machines go down, the other machine’s proxy takes over. This principle supports fault tolerance.

STRUCTURE

Overview

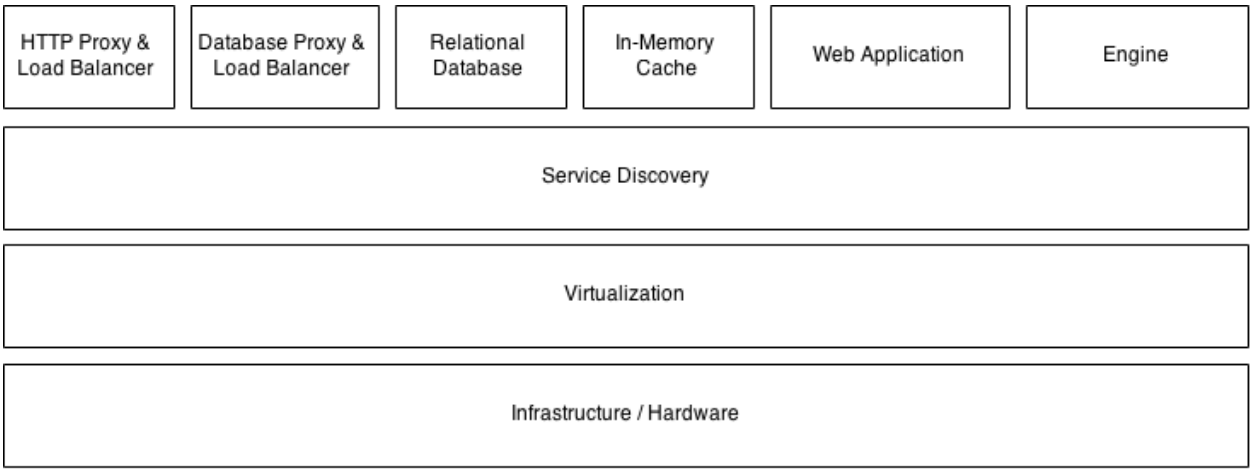

Figure 2: Overview of the architecture of the system

The tutoring system has 6 principal components. These are the HTTP proxy, database proxy, relational database, in-memory cache, web application and engine. All these components reside on top of a service discovery layer, which is responsible for keeping track of the health and presence of the instances of these components. Below the service discovery layer lies a virtualization layer, which enables creation and deletion of instances dynamically within a physical machine. It also enables distribution and relocation of instances across physical machines, without any changes to the instances themselves. The physical hardware forms the lowest layer of the architecture.

The virtualization technology used for the system is Linux Containers. Containers are a better approach to virtualization than standard hypervisors, owing to their lightweight nature. Docker has been chosen as the technology to containerize the application components. A Docker container is almost as light as running a process within an operating system. Also they are portable. You can read on Docker[20] containers to know about their advantages and disadvantages

The system is designed to scale horizontally. Thus we can expect the addition and removal of service instances to and from it, dynamically. This requires keeping track of what services are running and where they are running. Also components within the system, which need to interact with one another, need to know of each other in a dynamic environment. Thus, service discovery is used in the deployment of the system. Consul[21] has been chosen as the software

[20] dotCloud, “Docker,” n.d., https://www.docker.com/whatisdocker/.
[21] Hashicorp, “Consul,” n.d., https://www.consul.io/.

which acts as the service discovery agents in the various components across the system. Whenever a component boots up, it adds itself to the system cloud, by registering itself as a service on the system. Other components, which require the services offered by this component consults the service discovery agent to get the location and configuration of the service providing node. It then uses this information to communicate with the node. Whenever a component leaves the system, it deregisters itself from the service cloud and exits. This way, nodes can be dynamically added and removed from the system without affecting service availability. Another important feature of the service discovery agents is to keep track of the health of the individual nodes. By registering checks for itself on its agent, each node publishes its health status via the agent to the service cloud. This way, monitoring applications can keep track of what services are unavailable and should be replaced or repaired.

HTTP Proxy: This component acts as a proxy[22] as well as a load balancer[23] for all HTTP requests corresponding to the web services offered by the Engine and Web Application. Requests are redirected to either the Engine or the Web Application instances according to some URL rules specified in its access control list. It also load balances[24] the requests among all the instances of each of these two types of web servers. The least connected load balancing algorithm is used to distribute the requests. The proxy uses service discovery to keep track of what instances of each web service are available and healthy. It uses this information to update its server list whenever there is a change. This mechanism enables instances to be dynamically added and removed from the system without any downtime. HAProxy[25] is used as the load balancer.

Database Proxy: This component acts as a proxy and a load balancer for the database instances. The job of this component is to redirect requests for database accesses to one of the available database instances, based on the least connected load balancing algorithm. The proxy acquires information about the available database instances through a service discovery agent. It updates its server list based on the health and availability of such instances. HAProxy is used as the database proxy.

The rationale behind the requirement of a database proxy is to uniformly distribute database load across all the available database instances. It also helps to dynamically add and remove database instances from the system without any downtime. As database queries can vary based on the amount of time required to address them, it is not wise to statically assign database instances to application nodes which require it. This is because; a single application node can consume a lot of database resources, while others may be almost idle. Thus the entire load gets concentrated on a single database node, which is inefficient. A load balancer is kept to tackle this situation, which balances load among the running instances of database nodes.

---

[22] "Proxy Server," *Wikipedia*, n.d., http://en.wikipedia.org/wiki/Proxy_server.
[23] "Load Balancing (computing)," *Wikipedia*, n.d., http://en.wikipedia.org/wiki/Load_balancing_%28computing%29.
[24] Valeria Cardellini, Michele Colajanni, and Philip S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing*, 1999.
[25] HAProxy, "HAProxy," n.d., http://www.haproxy.org/.

Relational Database: This component is responsible for storing all data corresponding to the system. The data stored includes user accounting information, code history, programming problems and their test cases, course events and corresponding assignments, grading information, compilation logs, execution logs and evaluation logs. The database also stores various other information required for system operation such as configurations. This component is the sole persistent data store for the system. The database instances are clustered using replication, so that they together form a high-availability, fault-tolerant and durable data store. The cluster uses Galera[26] for master-master replication with synchronous writes. This implies that all database instances have the same data at any instant of time. This feature guarantees data consistency. MySQL[27] is used as the data store.

One of the main purposes of the system is to provide a framework for collection of data for analytics. Since the data collected by the system is relational in nature, a relational database[28] has been chosen as the data storage engine. Also a relational database supports querying in ways which support providing statistical information directly. This is necessary for various research activities which require such information.

In-Memory Cache: This component is responsible for session storage and database caching. Memcached[29] is used as the cache. Use of in-memory storage for sessions ensures high performance, where almost all of the requests for web services require authentication. Database caching is also necessary as a lot of database queries are expensive and do not change frequently. The cache instances are also clustered. However the data across the instances are sharded to optimize performance. The cache component has no persistence, and in the event of a node failure, all data stored in that node is lost. However, since the sort of data stored in the cache instances are not valuable and are recoverable, this is not a problem. The cache cluster also does not involve replication, so as to minimize the overhead involved.

Session stores mostly use persistence, in which they save the contents to the disk in either regular files or databases. While designing the session store, it was observed that almost all of the requests that are served by either of the services - engine or web application, require authentication. Thus from the perspective of the web applications, they must look up the session stores to fetch/write information on every request. For a session store that uses persistence, this is quite expensive and may be a bottleneck for the application itself. Thus an in-memory cache is used as a session store, to speed up the process. Memcached is used for this purpose as it can be sharded and distributed across nodes with ease. One may argue that using a persistent store with caching such as Redis[30] can be a better option, as it has persistence as well as in-memory caching. However, firstly clustering support for Redis is not yet stable and moreover

[26] Galera, “Galera,” n.d., http://galeracluster.com/products/technology/.
[27] Oracle, “MySQL,” *MySQL*, n.d., https://www.mysql.com/.
[28] T. Bakuya and M. Matsui, *Relational Database Management System* (Google Patents, 1997), https://www.google.com/patents/US5680614.
[29] Memcached, “Memcached,” n.d., http://memcached.org/.
[30] Redis, “Redis,” n.d., http://redis.io/.

writes to Redis will cause an overhead larger than Memcached. Further, as session data is not valuable data, persistence is not necessary. Losing session data simply results in the end user being logged out. We can always add redundancy to the Memcached cluster to avoid such losses, if necessary.

Web Application: This component is responsible for providing user interfaces and API services corresponding to the web application offered by the system. The operations performed by this component are mostly database and hence I/O intensive. Thus NodeJS[31], a web server with asynchronous I/O is chosen for this purpose. Each instance of this component consumes a single thread, and hence multiple instances are required to increase performance. Thus, the web application subsystem is clustered as well. However each node in the cluster is independent of one another. The HTTP proxy balances the load among these nodes. The web application subsystem has the largest codebase among all the components present in the system.

Engine: This component is responsible for handling all compilation, execution and evaluation requests of the tutoring system, along with requests for execution of some tools. It performs all of the major compute intensive operations of the system. The Engine uses a multi-threaded web server Apache[32], to handle its requests, so that each request can be assigned to a single thread. The web application is itself written in PHP[33]. This component is also clustered so that Engine nodes can be added and removed dynamically from the system. The HTTP proxy balances the requests among these nodes.

The engine is a web application which runs behind a web server. Since the time taken for the operations that the engine performs can be quite high, it is necessary to queue them. However, an explicit message queue is not maintained for this purpose. Instead the web server is configured to receive a maximum of *n* clients for processing by the engine. The value of *n* is carefully chosen such that the instance is able to process all the n requests simultaneously. Any more requests that come in during the time when the engine is saturated, get queued by the web server automatically. Thus a queue is implicitly maintained by the web server. Moreover the timeouts for the web server are chosen such that they exceed the maximum expected engine time for any request.

The services provided to the users are split between two components - the engine and the web application server. This is done intentionally, to enable scaling of the system based on usage of services and provisioning the components on different hardware based on the type of resources that they consume most. The web application is a database intensive piece of software, which in turn implies that it is an I/O intensive application. An application server which supports this sort of usage is Node JS. Thus the web application server is built using it. On the other hand, services which involve compilation and execution are CPU intensive and require an environment which supports this. Apache coupled with PHP is used for this purpose. Although it

[31] Joyent, “Node JS,” n.d., https://nodejs.org/.
[32] Apache Software Foundation, “Apache HTTP Server,” n.d., http://httpd.apache.org/.
[33] PHP, “PHP,” n.d., http://php.net/.

may be a good option to use “better” environments such as Java[34] for this purpose, this choice has been made from the perspective of rapid prototyping. Since Apache spawns new threads for each request, a separate thread can be dedicated for an engine request. Node JS on the other hand cannot support this as it is single threaded.

Static Content Proxy: Apart from the regular web and application servers, additional proxy servers are maintained in order to serve static content. This has not been included in the diagram above, as it is optional. The static content proxy server is used to relieve the application servers of unnecessary load, and also to speed up site loading in browsers by strategically placing static content on high speed web servers.

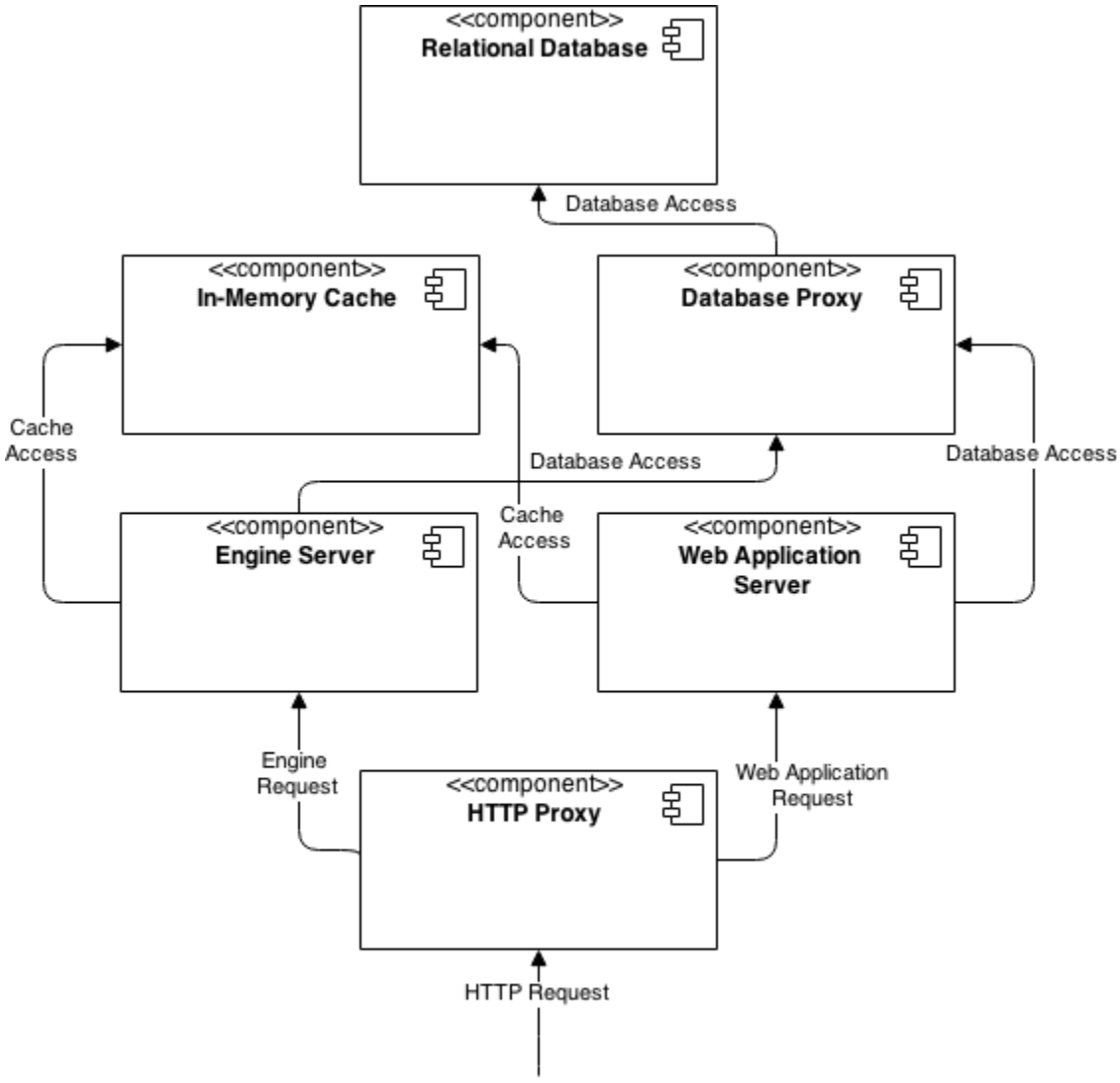

Figure 3: Components connectivity

All HTTP requests arrive at the HTTP proxies. The proxy servers then forward these requests to either the Web Application servers or the Engine servers, depending upon the URL path of the

[34] Oracle, “Java EE,” n.d., http://www.oracle.com/technetwork/java/javaee/overview/index.html.

request. Load balancing at the HTTP level happens at this step. Each of the Engine and Web Application servers requires the services of the Database and In-Memory Cache servers. The cache servers are accessed directly, whereas the database servers are accessed via the Database Proxy.

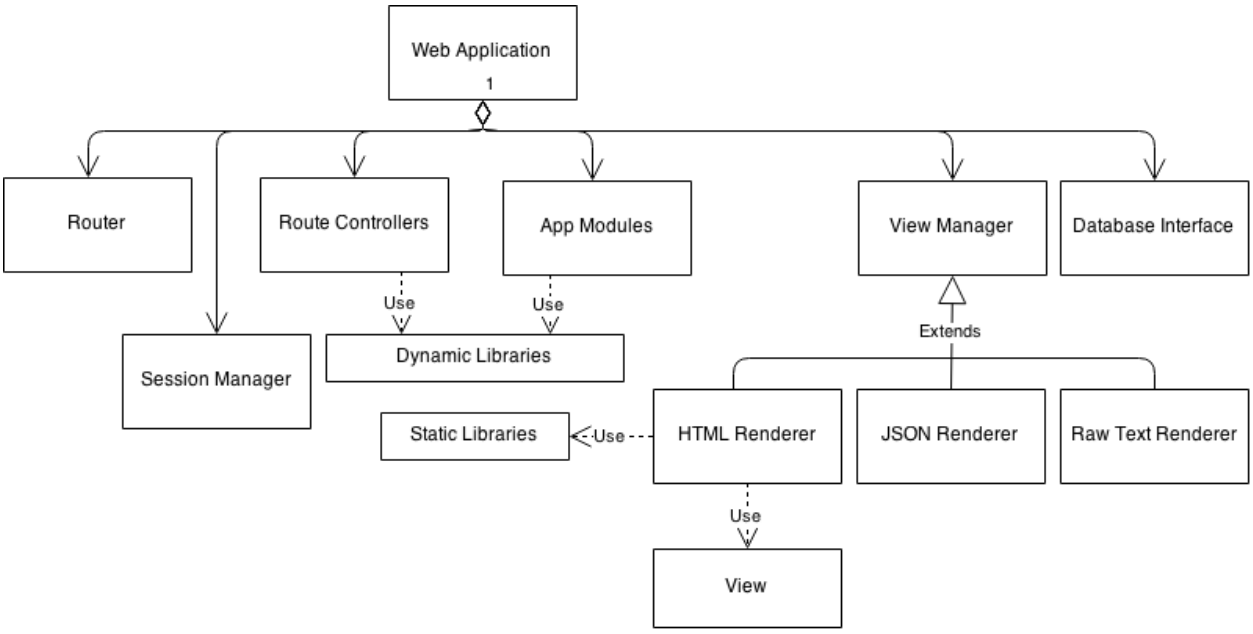

Figure 4: A logical view of the Web Application subsystem

The web application is composed of four basic components - Router, Route Controllers, App Modules and the View Manager. The Router is responsible for invoking the appropriate action whenever a web request comes in. Based on the HTTP request, it invokes the corresponding controller. It is also responsible for filtering out non-authentic requests. It uses the session manager to check whether the request belongs to a valid user session or not. It also checks whether the user belongs to the correct role required to access the requested resources. The controller is responsible for a group of routes which handle similar functionalities. It acts as a coordinator between the various application modules and the view manager. It invokes the required modules with the necessary arguments, based on the request and forwards the final output to the view manager for rendering to the client. The application modules handle specific sets of functions. They interact with the database of the system, using the database interface, to perform basic CRUD operations as well as some computations. The view manager is responsible for rendering output to the client, after all necessary data retrieval and computations have been performed. The view manager renders the output in one of the three formats - HTML, JSON or raw text. The HTML renderer is generally used for rendering user interfaces, whereas the JSON and raw text renderer are used to output data requested via API calls. The HTML renderer uses view definitions to render its output.

The web application uses a variation of the Model View Controller (MVC)[35] architecture. Each route has a controller, which can use multiple modules in the application.

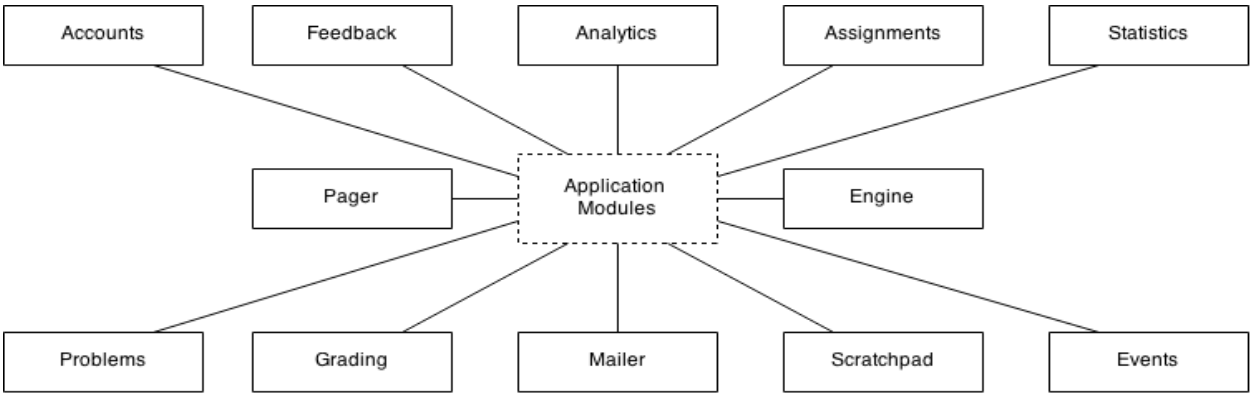

Figure 5: The application modules in the Web Application subsystem

The web application subsystem contains a number of modules which are responsible for delivering functionalities for various features of the system. The accounts module is responsible for managing user accounts on the system as well as handling user authentication. The problems module is responsible for managing programming problems and their test cases used in conducting the course. The events module manages the course events and their schedules. The statistics module is used to provide statistical information regarding the conducted course. Examples of statistical information include number of submitted solutions, number of correct solutions, number of labs conducted, etc. The feedback module is responsible for managing data for feedback tools, integrated into the system. The analytics module is used for generating analytics from the data collected by the system. The assignments module is responsible for managing the course assignments and maintaining code history for the assignments. The pager handles the messaging system within the application. The engine module is responsible for updating the engine configurations to the database, so that they can be synchronized across nodes. The grading module manages the grades of assignments for various course events. The mailer module is used to send email messages by the system. The scratchpad module handles the creation, deletion and modification of files in the student scratchpad.

[35] Glenn E Krasner, Stephen T Pope, and others, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System," *Journal of Object Oriented Programming* 1, no. 3 (1988): 26–49.

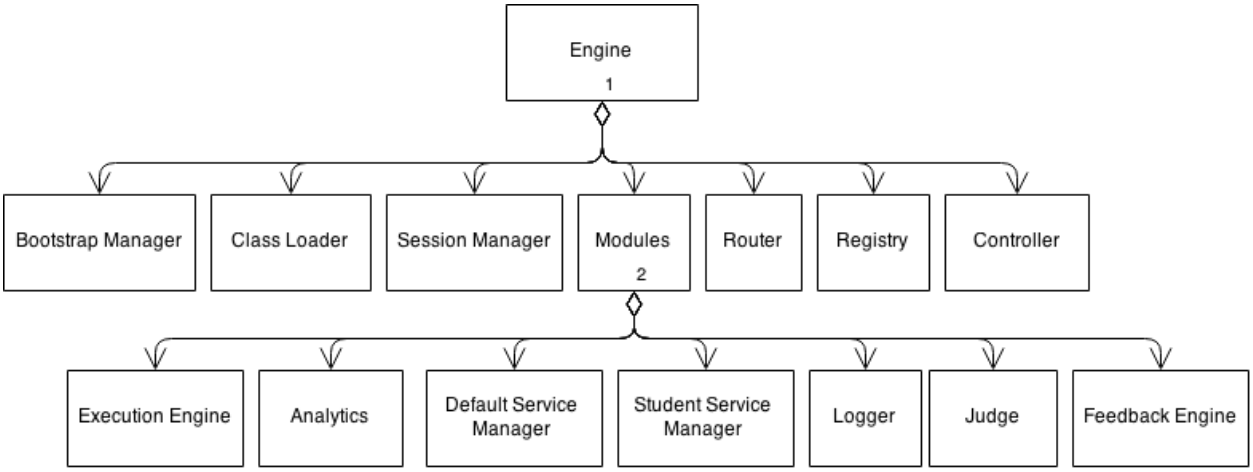

Figure 6: A logical view of the Engine subsystem

The engine is composed of six major components - bootstrap manager, session manager, application modules, router, registry and controller. The bootstrap manager boots the sub-system on every request, so as to include the necessary classes and methods for use in the later stages. Only the necessary classes and modules are loaded by the bootstrap manager, which includes the class loader. The class loader loads the required classes dynamically on each request. The session manager uses the cookie information provided with the request to load the user session into the request. Once the bootstrap manager finishes booting, it invokes the router with the request path and parameters. The router then decides the action to be performed based on the request path, and invokes the appropriate controller for the same. The router uses the registry to decide which controller to invoke, based on a mapping created previously. The controller coordinates between the application modules to perform the required action, as specified by the request. It creates objects of the necessary modules, once it gets the request from the router, and invokes the appropriate methods on them. It also uses the session manager to decide if the request is authentic or not and drops the request with an error or redirection if it isn't. The application modules are not completely decoupled, and use methods from other modules to achieve their purposes. The final result received by the controller from the application modules is then marshalled into a JSON string and sent back to the client.

The modules in the engine subsystem deliver the functionalities offered by it. The execution engine is responsible for compiling and executing programs. The analytics module is used to perform analysis on data collected by the system. The default service manager handles engine requests by admin users and students using the system for purposes other than course events. The student service manager handles requests by students using the system for course events. The logger logs the results of compilation, execution and evaluation for course events. The judge is responsible for evaluating student programs. The feedback engine generates feedback for the student programs.

The engine has a variation of the Model View Controller (MVC) architecture. Here, the controller does not have a dedicated model to itself. Instead, it can communicate with multiple modules to achieve its purpose. This promotes reuse of modules within the system.

Components

<table>
<tr><td>Component</td><td>Engine</td></tr>
<tr><td>Responsibilities</td><td>Compiles, executes and evaluates programs. It also runs various tools integrated into the system. Examples of tools are feedback generation tools.

The provided interfaces are:
1. Engine endpoint.</td></tr>
<tr><td>Collaborators</td><td><table><tr><th>interface</th><th>component</th></tr><tr><td>Database proxy</td><td>Relational Database</td></tr><tr><td>Cache interface</td><td>In-Memory Cache</td></tr></table></td></tr>
<tr><td>Notes</td><td>Uses the database to read configurations. It is dynamically created and destroyed. It is created when an engine node is spawned. It is destroyed when the node is destroyed. A single engine instance uses multiple threads.</td></tr>
<tr><td>Issues</td><td>Needs to handle multiple programming languages and paradigms.</td></tr>
</table>

<table>
<tr><td>Component</td><td>Web Application</td></tr>
<tr><td>Responsibilities</td><td>Responsible for rendering user interfaces for the tutoring system. Handles API requests corresponding to the web application, such as code saves, fetching grade cards, fetching assignment problems, etc. Also responsible for creating and destroying user sessions for the application.

The provided interfaces are:
1. Web application interface
2. User interfaces</td></tr>
<tr><td>Collaborators</td><td><table><tr><th>interface</th><th>component</th></tr><tr><td>Database proxy</td><td>Relational</td></tr></table></td></tr>
</table>

	<table><tr><td></td><td>Database</td></tr><tr><td>Cache interface</td><td>In-Memory Cache</td></tr></table>
Notes	Persistent for the lifetime of the node on which it runs. Destroyed when the node is destroyed. An instance is single threaded.
Issues	Error handling not stable.

Component	In-Memory Cache
Responsibilities	Stores session information and database rows for caching. The provided interfaces are: 1. Cache interface
Collaborators	
Notes	Created when a cache node is created and destroyed when it is destroyed.
Issues	Sharding algorithms used by the engine and web application nodes do not match perfectly.

Component	Relational Database
Responsibilities	Stores all persistent data for the system and collected by the web services. The provided interfaces are: 1. Database interface
Collaborators	
Notes	Created when a database node is created and destroyed when it is destroyed. Uses multiple threads.
Issues	Clustering uses synchronous writes. Hence writes are slow.

Component	Database Proxy
Responsibilities	Load balances database requests among the available database nodes.

<table>
<tr><td></td><td>The provided interfaces are:
1. Database proxy interface</td></tr>
<tr><td>Collaborators</td><td><table><tr><th>interface</th><th>component</th></tr><tr><td>Database interface</td><td>Relational Database</td></tr></table></td></tr>
<tr><td>Notes</td><td>Created when a new system is deployed. Persists throughout the lifetime of the system. Destroyed only when replaced.</td></tr>
<tr><td>Issues</td><td>Connection timeouts need to be inferred from system performance.</td></tr>
</table>

<table>
<tr><td>Component</td><td>HTTP Proxy</td></tr>
<tr><td>Responsibilities</td><td>Load balances HTTP requests among the Web Application and Engine nodes.

The provided interfaces are:
1. HTTP proxy interface</td></tr>
<tr><td>Collaborators</td><td><table><tr><th>interface</th><th>component</th></tr><tr><td>Web Application Interface</td><td>Web Application</td></tr><tr><td>Engine endpoint</td><td>Engine</td></tr></table></td></tr>
<tr><td>Notes</td><td>Created when a new proxy node is created. Destroyed when it is destroyed. Usually remains throughout the lifetime of the system.</td></tr>
<tr><td>Issues</td><td>Load balancing weights need to be determined based on the availability of web application and engine nodes.</td></tr>
</table>

Interfaces

Interface	Web application interface
Description	Provides access to the Web Application services.
Services	**Operation:** render login page **Description:** Renders the user interface to enable users to log into the system. **Operation:** login **Description:** Attempts to login a user to the system. **Operation:** logout **Description:** Logs out an user from the system and clears session information. **Operation:** render home page **Description:** Renders the home page for a student. **Operation:** render event editor interface **Description:** Renders the editor interface corresponding to a course event. **Operation:** render scratchpad interface **Description:** Renders the editor interface for the student scratchpad. **Operation:** render practice arena **Description:** Renders the user interface containing practice problems. **Operation:** render codebook **Description:** Renders the user interface corresponding to a student and containing all the code submissions for course events as well as practice problems. **Operation:** render codebook page **Description:** Renders the user interface containing a code view, grading information and problem statement corresponding to a

	student's codebook entry. **Operation:** save code for assignment **Description:** Saves a code version for an assignment or practice problem, in a certain mode. **Operation:** create pager message **Description:** Creates a new message on the pager feature, corresponding to a certain context. **Operation:** respond to pager message **Description:** Creates a response for a pager message thread. **Operation:** delete pager message **Description:** Deletes a message corresponding to a pager thread. **Operation:** render pager view **Description:** Renders the user interface containing all message threads created by the student. **Operation:** create a file **Description:** Creates a virtual scratchpad file on the system. **Operation:** delete a file **Description:** Deletes a scratchpad file from the system. **Operation:** save scratchpad file **Description:** Saves the contents of a scratchpad file to the database. **Operation:** create a folder **Description:** Creates a virtual ScratchPad folder on the system. **Operation:** render admin home **Description:** Renders the admin home user interface, from where various admin services can be accessed. **Operation:** render user accounts **Description:** Renders the user interface listing the user accounts on the system.

	Operation: create admin account **Description:** Creates a new admin user account on the system. **Operation:** create student account **Description:** Creates a new student user account on the system. **Operation:** delete admin account **Description:** Deletes an admin user account from the system. **Operation:** delete student account **Description:** Deletes a student user account from the system. **Operation:** modify admin role **Description:** Modifies the role taken by an admin on the system, such as an instructor, tutor or teaching assistant. **Operation:** modify admin name **Description:** Modifies the name by which the admin user is identified on the system. **Operation:** modify admin password **Description:** Modifies the password for an admin user. **Operation:** modify student account **Description:** Modifies profile and accounting information related to a student user. **Operation:** render problem management portal **Description:** Renders the user interface for managing problems for the course which is being run using the system. **Operation:** render problem view **Description:** Renders an interface containing all information related to a problem, and where the problem can be edited. **Operation:** render problem upload **Description:** Renders the interface using which problems can be uploaded to the system in batch. **Operation:** update problem statement

	Description: Updates the problem statement for a problem instance. **Operation:** update problem solution **Description:** Updates the solution code for a problem. **Operation:** update problem template **Description:** Updates the initial template corresponding to a problem. **Operation:** update problem title **Description:** Updates the title corresponding to a problem. **Operation:** add test case **Description:** Adds a test case for a specific problem. **Operation:** remove test case **Description:** Removes a test case from the system. **Operation:** add bulk test cases **Description:** Adds test cases for a problem in batch. **Operation:** delete problem **Description:** Deletes a problem from the system. **Operation:** mark problem practice **Description:** Marks a specific problem as a practice problem. **Operation:** render event management portal **Description:** Renders the user interface using which course events can be added, schedules can be made and problems can be assigned to students. **Operation:** create event **Description:** Creates a course event on the system for the on-going course. **Operation:** delete event **Description:** Deletes a previously created event from the system. **Operation:** schedule event

	Description: Creates a schedule for a particular event. **Operation:** assign problems **Description:** Assign problems for a course event to students based on some algorithm. **Operation:** add slots for schedule **Description:** Add slots for a specific schedule of an event. Slots define which students correspond to a particular schedule. **Operation:** render admin tasks portal **Description:** Render the user interface showing the pending tasks for an admin as well as all pending tasks grouped by respective course events. **Operation:** render submissions view **Description:** Render the user interface containing student submissions to programming assignments. **Operation:** render code viewer **Description:** Renders the user interface for viewing code history and grading. **Operation:** grade submission **Description:** Sets the grade for a programming assignment submission for the course. **Operation:** render assignment analytics **Description:** Renders the user interface containing analytics for a particular assignment. **Operation:** render admin editor **Description:** Renders the user interface from which admin users can compile, execute and evaluate arbitrary code. The editor is also used to update solution codes to problems. **Operation:** render control panel **Description:** Renders the user interface which is used to modify settings for the tutoring system such as execution delays and compiler flags. **Operation:** render admin settings

	Description: Renders the user interface from where admin users can modify their name and password. **Operation:** render admin pager **Description:** Renders the user interface using which admin users will be able to view message threads created by students.
Protocol	All activities using this interface must be preceded by a login.
Notes	Exposed as a web service.
Issues	Error handling not stable. Clean-up required in the codebase.

Interface	Engine endpoint
Description	Provides access to the Engine services.
Services	**Operation:** tool() **Description:** Runs a tool specified in the set of parameters sent to the server. Additional parameters may also be passed to this method, as required by it. **Operation:** compile() **Description:** Compiles a program, which is specified as a parameter and returns the result of compilation. This method may also log results, based on the context of the request. **Operation:** execute() **Description:** Executes a program on a test case as specified in the set of parameters, and returns the result of execution. This method may also log the results based on the context of the request. **Operation:** evaluate() **Description:** Evaluates a programming assignment submission on a set of test cases that were assigned to the problem, and returns the result. This method may log results depending on the context in which it was invoked.

Protocol	Compilation followed by execution or evaluation.
Notes	Exposed as a HTTP web service.
Issues	Sequence of requests is to be enforced.

Interface	Cache interface
Description	Provides access to the In-Memory cache.
Services	**Operation:** get key **Description:** Fetches the value corresponding to a key from the cache. **Operation:** set key **Description:** Sets a key along with its value, in the cache.
Protocol	There are no restrictions on accessing this interface.
Notes	A single interface is exported by a single cache node.
Issues	No issues currently.

Interface	Database node interface
Description	Provides access the database server.
Services	**Operation:** run SQL queries **Description:** Runs SQL queries against the database server hosted on that node.
Protocol	There are no restrictions on accessing this interface.
Notes	A database node exports a single interface for it.
Issues	No issues currently.

DYNAMIC BEHAVIOUR

Scenarios

Scenario Specification

Use Case	ViewGradesAndStatistics
Description	View grades for assignments and course statistics.
Actors	Student
Steps	1. User logs in to the system. 2. System redirects user to the home page. 3. User views grading information and statistics related to the course.

Use Case	ViewOngoingEvents
Description	View the on-going event along with a summary of the problems assigned to the student, if any.
Actors	Student
Steps	1. User logs in to the system. 2. Systems redirects user to the home page. 3. User views any running on-going events in a panel along with a summary of the event.

Use Case	ViewAssignmentProblemStatement
Description	Views the programming problem for an assignment.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment

	problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student views problem statement in the problem tab in this user interface.

Use Case	WriteAssignmentProgram
Description	Writes a solution to a programming problem.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student writes programs in the supplied editor.

Use Case	CompileAssignmentProgram
Description	Compiles a program written for a programming assignment.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student writes program in editor. 7. Student compiles program by clicking on the compile button in the menu bar or hitting the shortcut key for compile.

Use Case	ExecuteAssignmentProgram
Description	Executes a program written for a programming assignment.

Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student writes program in editor. 7. Student executes program by clicking on the execute button in the menu bar or hitting the shortcut key for execute.

Use Case	EvaluateAssignmentProgram
Description	Evaluates a program against a set of test cases for a programming assignment.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student writes program in editor. 7. Student evaluates program by clicking on the evaluate button in the menu bar or hitting the shortcut key for evaluate.

Use Case	SubmitAssignment
Description	Submits a program as a solution to an assignment problem.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to the home page. 3. Student views on-going event. 4. Student clicks on an assignment

	problem displayed in the summary. 5. Student is redirected to “Editor for course events” interface. 6. Student writes program in editor. 7. Student submits the program by clicking on the submit button in the menu bar.

Use Case	CreateScratchPadFileFolder
Description	Creates a virtual file in the scratchpad interface.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on scratchpad link. 4. System redirects student to Scratchpad interface. 5. Student creates a virtual file or folder by clicking on “create file” or “create folder” from the menu bar or using the context menu invoked by right clicking on the workspace area.

Use Case	DeleteScratchpadFileFolder
Description	Deletes a virtual file or folder from the scratchpad.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on scratchpad link. 4. System redirects student to Scratchpad interface. 5. Student deletes a virtual file or folder by clicking on “delete file” or “delete folder” from the menu bar or using the context menu invoked by right clicking on the workspace area.

Use Case	WriteScratchProgram

Description	Writes a program for learning or experimental purposes or for any other purpose than solving a programming assignment.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on scratchpad link. 4. System redirects student to Scratchpad interface. 5. Student creates and/or opens a virtual file. 6. Student writes his program into it.

Use Case	CompileScratchProgram
Description	Compiles a scratch program previously written by a student.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on scratchpad link. 4. System redirects student to Scratchpad interface. 5. Student creates and/or opens a virtual file. 6. Student writes a program. 7. Student compiles the program by clicking on the compile button in the menu bar or by hitting the corresponding shortcut key.

Use Case	ExecuteScratchProgram
Description	Executes a scratch program previously written by a student.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on scratchpad link. 4. System redirects student to

	Scratchpad interface. 5. Student creates and/or opens a virtual file. 6. Student writes a program. 7. Student executes the program by clicking on the execute button in the menu bar or by hitting the corresponding shortcut key.

Use Case	ViewAttemptedProblems
Description	Views problems attempted by student in course events and practice arena.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on CodeBook link. 4. System redirects student to Codebook interface. 5. Student views list of attempted problems. 6. Student clicks on an item in the list. 7. System displays codebook page view. 8. Student views attempted problem and her submitted solution.

Use Case	ViewPracticeProblems
Description	Views practice problems uploaded by the instructor or tutors.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on Practice Arena link. 4. Systems redirects student to Practice arena interface. 5. Student views practice problem list. 6. Student clicks on an item in the list. 7. System displays practice problem. 8. Student views practice problem.

Use Case	CreateMessageThread
Description	Creates a message thread to obtain support from admins.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on on-going event problem. 4. System redirects student to Event editor interface. 5. Student clicks on support button from menu bar. 6. System displays popup window for writing a message. 7. Student writes message. 8. Student clicks on send. 9. System creates support thread.

Use Case	RespondToMessage
Description	Responds to a reply from an admin on a message thread previously created by the student.
Actors	Student
Steps	1. Student logs in to the system. 2. System redirects student to home page. 3. Student clicks on Pager link. 4. System redirects student to Pager interface. 5. Student views replies by admins on her message thread. 6. Student writes a response to the replies. 7. Student clicks on reply button. 8. System creates a response to the replies.

Use Case	CreateProblem
Description	Creates a problem to be used later on in the course.

Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on problem management link. 4. System redirects admin to Problem management user interface. 5. Admin enters an identifier for the problem along with a category in a provided box. 6. Admin clicks on create. 7. System creates a blank instance of a problem.

Use Case	EditProblem
Description	Edits a problem in the database.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on problem management link. 4. System redirects admin to Problem management user interface. 5. Admin clicks on problem from list. 6. System displays problem details. 7. Admin edits components of the problem such as statement and solution. 8. Admin clicks on save. 9. System saves problem.

Use Case	AddTestCase
Description	Adds a test case for a problem.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin

	home page. 3. Admin clicks on problem management link. 4. System redirects admin to Problem management user interface. 5. Admin clicks on problem from list. 6. System displays problem details. 7. Admin enters input in the input window for a new test case. 8. Admin clicks on save. 9. System generates output for the corresponding input, against the solution code to the problem. 10. System saves the test case.

Use Case	RemoveTestCase
Description	Deletes a test case corresponding to a problem, from the system.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on problem management link. 4. System redirects admin to Problem management user interface. 5. Admin clicks on problem from list. 6. System displays problem details. 7. Admin clicks on the delete icon beside an existing test case. 8. System deletes the test case from the system.

Use Case	UploadBulkProblems
Description	Uploads problems in batch to the system.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on problem management

	link. 4. System redirects admin to Problem management user interface. 5. Admin clicks on upload problems link. 6. System redirects admin to the upload problems page. 7. Admin clicks on select files link on the interface. 8. System opens up a file browser window. 9. Admin selects the files corresponding to the problems that he wants to upload and clicks on ok. 10. System inspects the set of files to check whether they are consistent with the required formats. 11. System uploads the files corresponding to all the problems that are consistent with the formats.

Use Case	AddAdminAccount
Description	Adds a user account for an admin user.
Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on the accounts management link. 4. System redirects admin to the accounts management portal. 5. Admin clicks on admin tab. 6. Admin clicks on add button inside the panels corresponding to one of the admin roles (instructor/tutor/teaching assistant). 7. System displays a form. 8. Admin enters respective details for new admin user. 9. Admin clicks on save. 10. System adds a new admin user to the system.

Use Case	AddStudentAccount
Description	Adds a user account for a student user.

Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on the accounts management link. 4. System redirects admin to the accounts management portal. 5. Admin clicks on student tab. 6. Admin enters email ID of the student he wants to add. 7. Admin clicks on add. 8. System adds a new student user with the corresponding email ID.

Use Case	DeleteAdminAccount
Description	Deletes a user account corresponding to an admin user.
Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on the accounts management link. 4. System redirects admin to the accounts management portal. 5. Admin clicks on admin tab. 6. System displays the list of admin users. 7. Admin clicks on the delete icon beside the name of an admin user. 8. System shows confirmation dialog for deletion of user. 9. Admin clicks on ok. 10. System deletes admin user account.

Use Case	EditStudentAccount
Description	Edits the account details corresponding to a student user account.
Actors	Instructor
Steps	1. Admin logs in to the system.

	2. System redirects admin to admin home page. 3. Admin clicks on the accounts management link. 4. System redirects admin to the accounts management portal. 5. Admin clicks on student tab. 6. System displays list of student users. 7. Admin clicks on edit button in the row corresponding to a student user. 8. System displays a dialog containing a form to edit the student's details. 9. Admin edits the necessary details. 10. Admin clicks on update. 11. System saves the updates to the student's account information.

Use Case	ViewCourseEventCalendar
Description	Views a calendar containing all the scheduled course events.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on event management link. 4. System redirects admin to event management portal. 5. Admin views the event calendar.

Use Case	CreateCourseEvent
Description	Creates a new course event.
Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on event management link. 4. System redirects admin to event management portal. 5. Admin clicks on add event button.

	6. System displays a form in which the admin can enter details corresponding to the new event and its schedule. 7. Admin enters necessary event details. 8. Admin clicks on create event. 9. System creates event and updates the calendar.

Use Case	AssignProblemsForEvent
Description	Assigns problems to students for a course event.
Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on event management link. 4. System redirects admin to event management portal. 5. Admin clicks on assign problems button. 6. System displays a wizard to specify the assignment algorithm. 7. Admin completes the wizard. 8. System creates required assignments for the course event.

Use Case	ViewEventDashboard
Description	View performance rankings of students for a course event along with a distribution of the scores used as the metric.
Actors	Instructor (primary) Tutor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on event management link. 4. System redirects admin to event management portal. 5. Admin clicks on dashboard. 6. System redirects admin to dashboard

	interface. 7. Admin views dashboard.

Use Case	ViewStudentSubmissionList
Description	Views the student submissions for a course event.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on student submissions link. 4. System redirects admin to student submissions page. 5. Admin views a list of student submissions, along with other meta information such as grading status.

Use Case	ViewCodeHistory
Description	Views the code history for a particular student submission.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on student submissions link. 4. System redirects admin to student submissions page. 5. Admin clicks on a submission link corresponding to an assignment. 6. System redirects admin to code viewer interface. 7. Admin views history of code saves corresponding to the assignment.

Use Case	GradeAssingmentSubmission

Description	Grades an assignment submission.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on student submissions link. 4. System redirects admin to student submissions page. 5. Admin clicks on a submission link corresponding to an assignment. 6. System redirects admin to code viewer interface. 7. Admin enters grade for the assignment in the respective box in the user interface. 8. Admin clicks on grade. 9. System saves the grade for the respective assignment.

Use Case	EvaluateStudentSubmission
Description	Evaluates a student submission for an assignment.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on student submissions link. 4. System redirects admin to student submissions page. 5. Admin clicks on a submission link corresponding to an assignment. 6. System redirects admin to code viewer interface. 7. Admin clicks on evaluate button. 8. System displays the result of evaluation of the submission corresponding to the assignment.

Use Case	ViewAssignmentAnalytics
Description	Views the analytics corresponding to a course assignment.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on student submissions link. 4. System redirects admin to student submissions page. 5. Admin clicks on a submission link corresponding to an assignment. 6. System redirects admin to code viewer interface. 7. Admin clicks on assignment analytics link. 8. System redirects admin to assignment analytics portal. 9. Admin views analytics corresponding to the assignment.

Use Case	ViewDataVisualizations
Description	Views data visualizations for data collected by the system.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on dataviz link. 4. System redirects admin to DataViz portal. 5. Admin views the visualizations created by developers, on the data collected by the system.

Use Case	ModifyEngineSettings

Description	Modifies various settings for the engine such as delays and quotas.
Actors	Instructor
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on control panel link. 4. System redirects admin to control panel interface. 5. Admin modifies values for various settings used by the engine. 6. Admin clicks on save. 7. System saves the settings.

Use Case	ModifyAccountSettings
Description	Modifies the name and password for the admin user account.
Actors	Instructor (primary) Tutor Teaching Assistant
Steps	1. Admin logs in to the system. 2. System redirects admin to admin home page. 3. Admin clicks on settings link in the menu bar. 4. System displays settings page to admin. 5. Admin updates the required information. 6. Admin clicks on update. 7. System updates the settings for the account.

Component Interaction Model

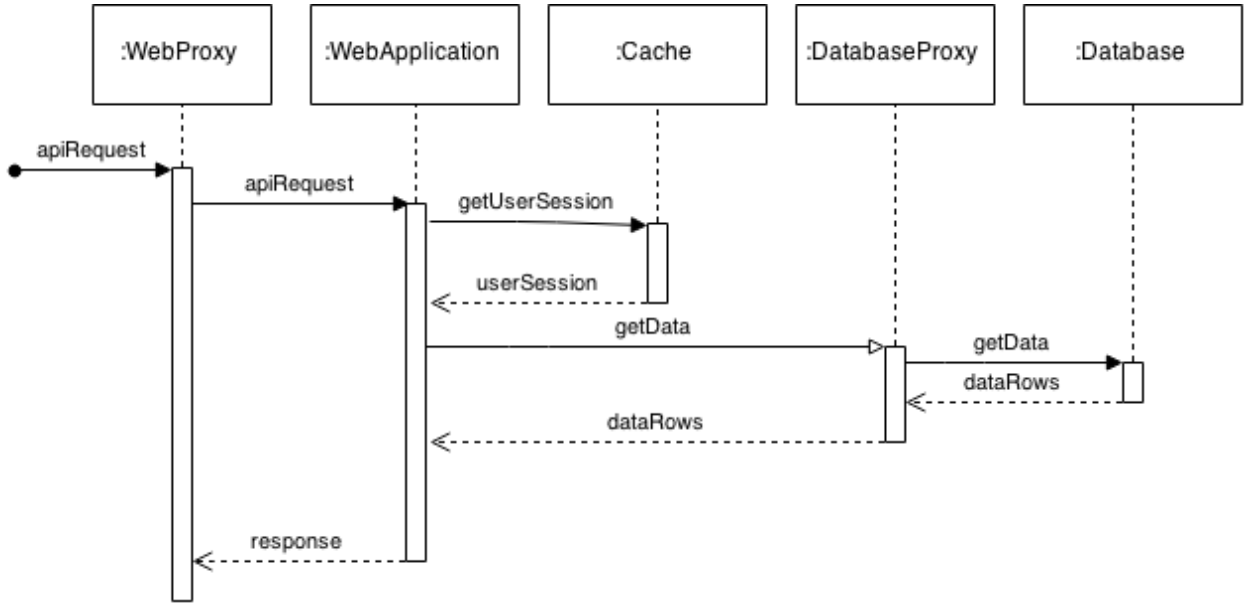

Figure 7: Sequence diagram for a Web API Request

The above diagram represents the flow of a basic API request from the client. The request first reaches the HTTP proxy and load balancer, which forwards it to one of the web application nodes based on its load balancing algorithm. The web application node, upon receiving it first checks whether the request is authentic or not. It does so by using the cookie information supplied with the request and consulting with the in-memory cache to fetch the corresponding session data. If no such data is present, then the request is not authentic. A request may also not be authentic, if the roles as specified in the session data, do not match with the role required the access the service. Once the request is authenticated, the appropriate modules are invoked and database accesses are made via the database proxy. The database proxy forwards these requests to one of the database nodes based on the load balancing algorithm used. Once the required result is available from the modules, it is sent back to the client.

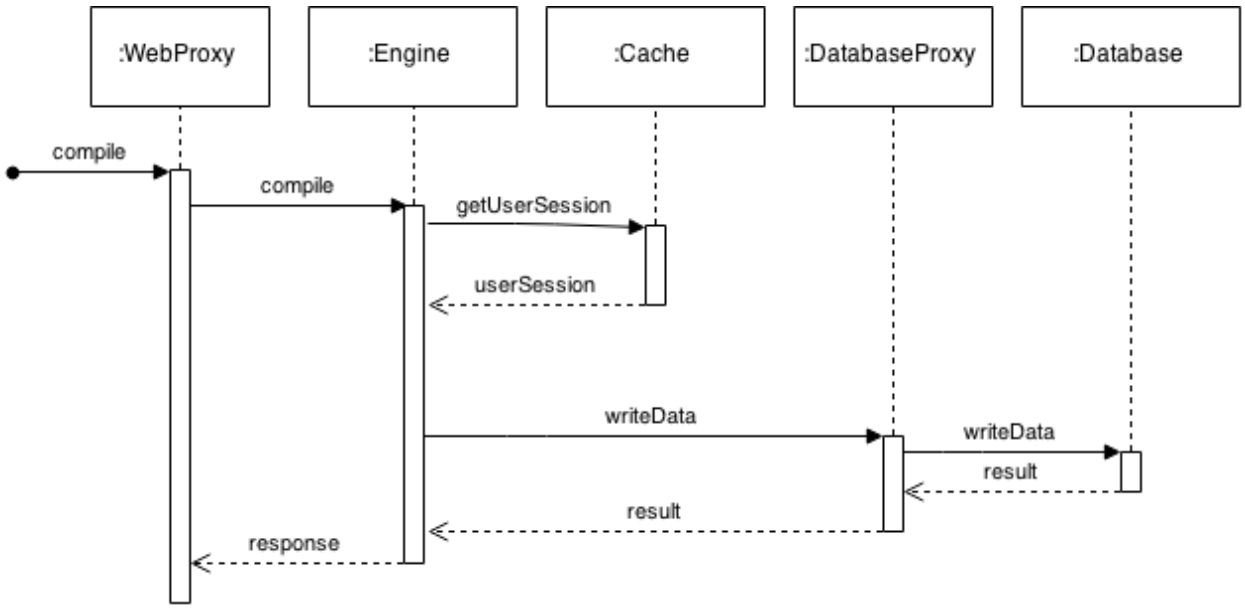

Figure 8: Sequence diagram for a compilation request

The above diagram represents the flow of a basic engine request. All engine service requests arrive at the web proxy which forwards them to an engine node. The engine then uses its session manager to look up the cache and verify the authenticity of the request. After verifying the authenticity, the corresponding tools for the request are invoked in the engine and database accesses are made via the database proxy. The final results of execution of required tools are sent as a marshalled JSON string, to the client.

Mechanisms

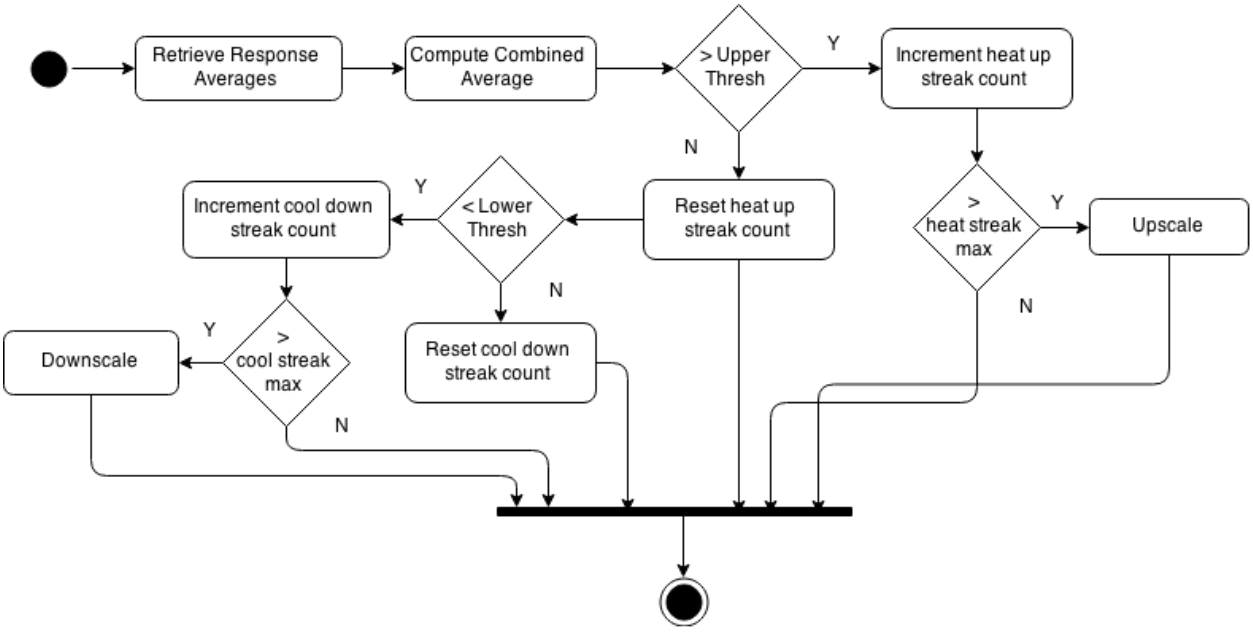

Figure 9: Activity diagram of scaling monitor for auto scaling web services

Auto Scaling: For web application nodes, the response time is logged by the application server. This log is used to compute the average response time over last n requests. This average response time, is published to a distributed key-value store at regular intervals. A monitor client for the host machine, which resides on the host machine itself, keeps track of these published response time entries. It first gathers information regarding the running containers on the host machine, and then computes the average over the response times published by each of the individual nodes on the key-value store. This average is used as a metric for the load on the web application servers. Once this value goes high, and stays high for a specific number of intervals, an upscale is triggered. This situation is called heat-up. The number of intervals for which the cumulative average stays high is referred to as the streak length. Once upscale is triggered, a new instance of the web application is spawned and connected to the cluster and load balancer. Similarly, if the average across all the web application nodes falls below a certain threshold and stays as such for a specific streak length, cooldown occurs, and a downscale is triggered. Downscaling simply removes a running instance from the cluster of web applications. The streak length for heat-up is much smaller than the streak length for cooldown (typically 1/10th). The time taken to provision a new instance is quite small (order of a few seconds). The idea of auto scaling is adapted from Scalr[36].

[36] Thomas Orozco, “Scalr Auto Scaling Algorithm,” n.d., https://scalr-wiki.atlassian.net/wiki/display/docs/Autoscaling+Algorithms.

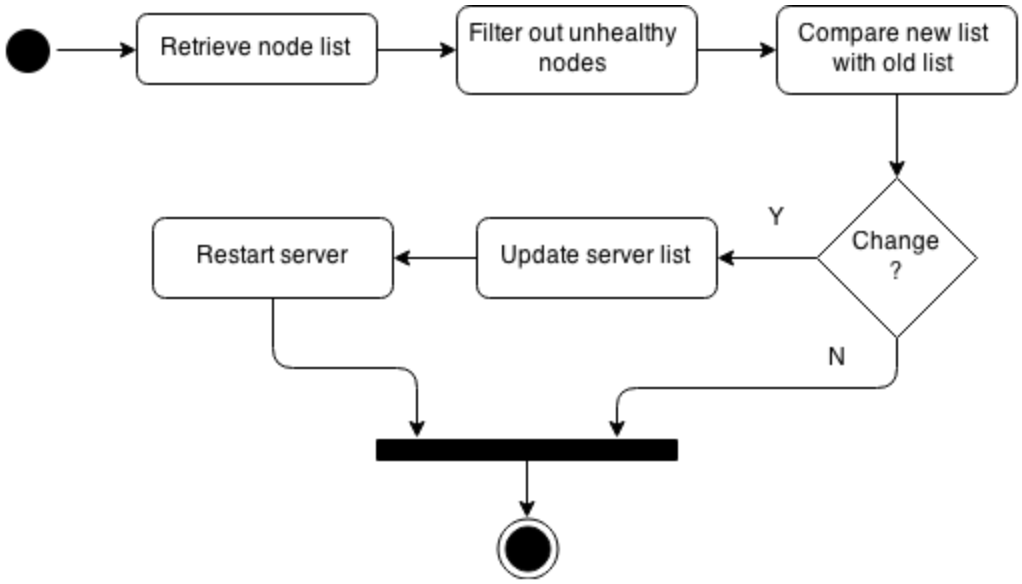

Figure 10: Activity diagram of load balancing monitor for dynamic load balancing

Dynamic Load Balancing: Due to the scalable nature of the system, nodes are added and removed from it dynamically. For the load balancer, this requires keeping track of what web service nodes are added to the system and what are removed. The load balancer forwards all HTTP requests to one of the web application or engine servers, based on the load balancing algorithm that it uses. However, in order to forward these requests, it must know the set of servers that are available. One cannot statically assign such servers to its server list as these can change arbitrarily. Thus the load balancer uses the services of the service discovery agent, to acquire information about what nodes are available and healthy. It then uses this information to dynamically update its server list, whenever there is a change.

OTHER VIEWS

Process View

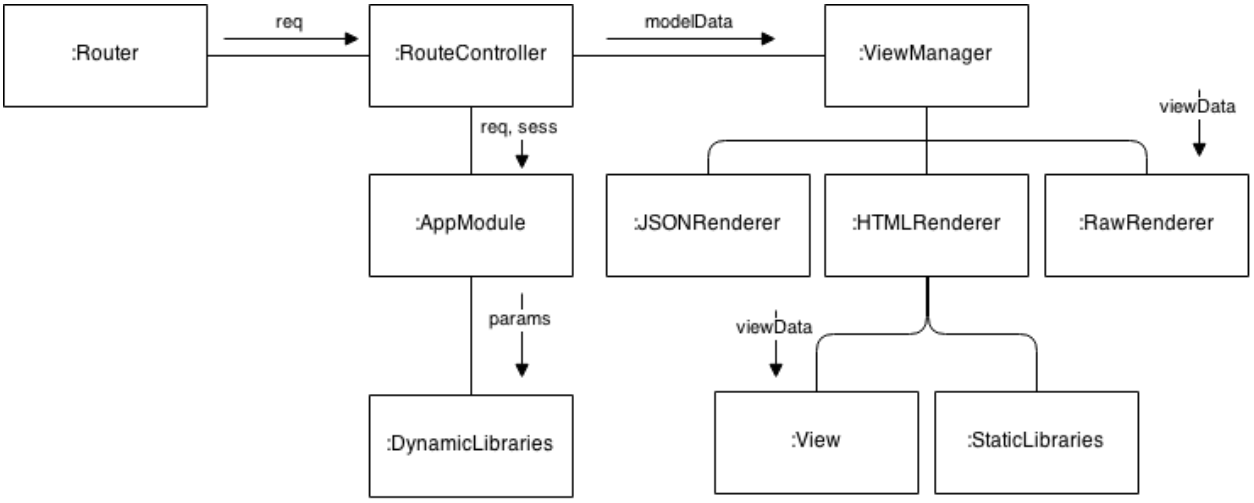

Figure 11: Collaboration diagram for Web Application subsystem

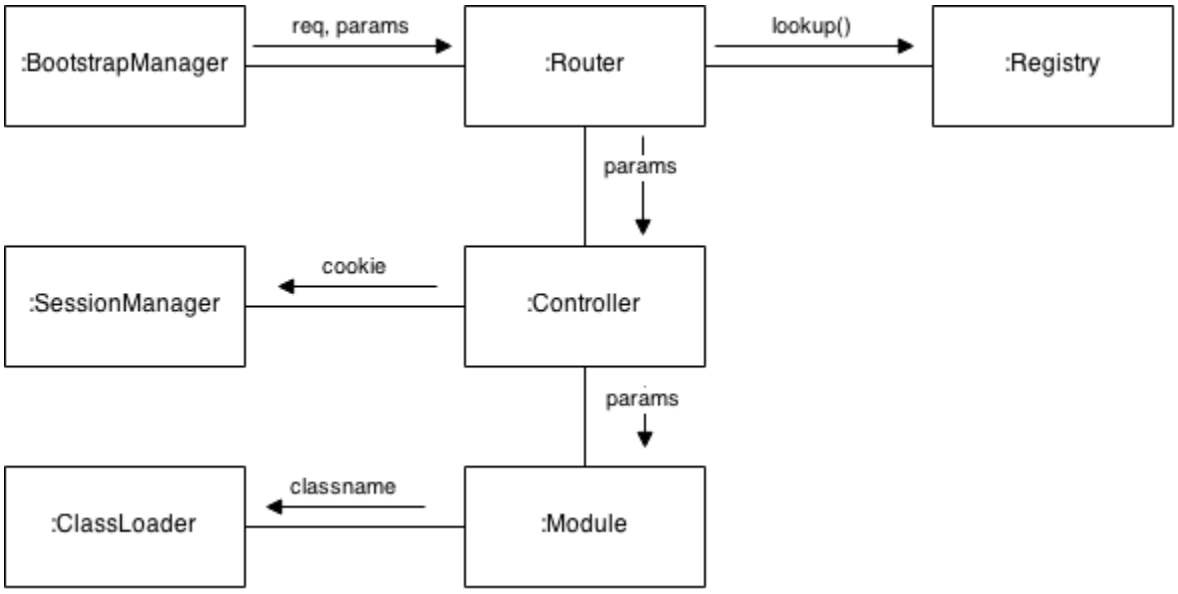

Figure 12: Collaboration diagram for Engine subsystem

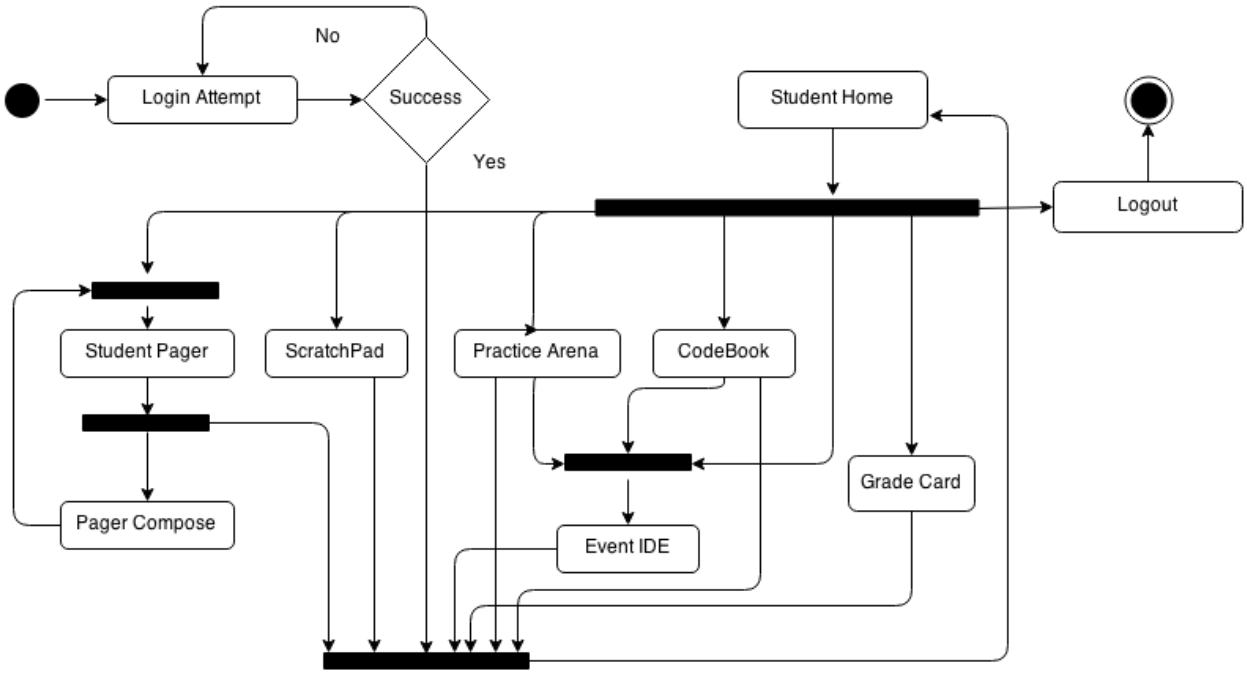

Figure 13: Normal student activity

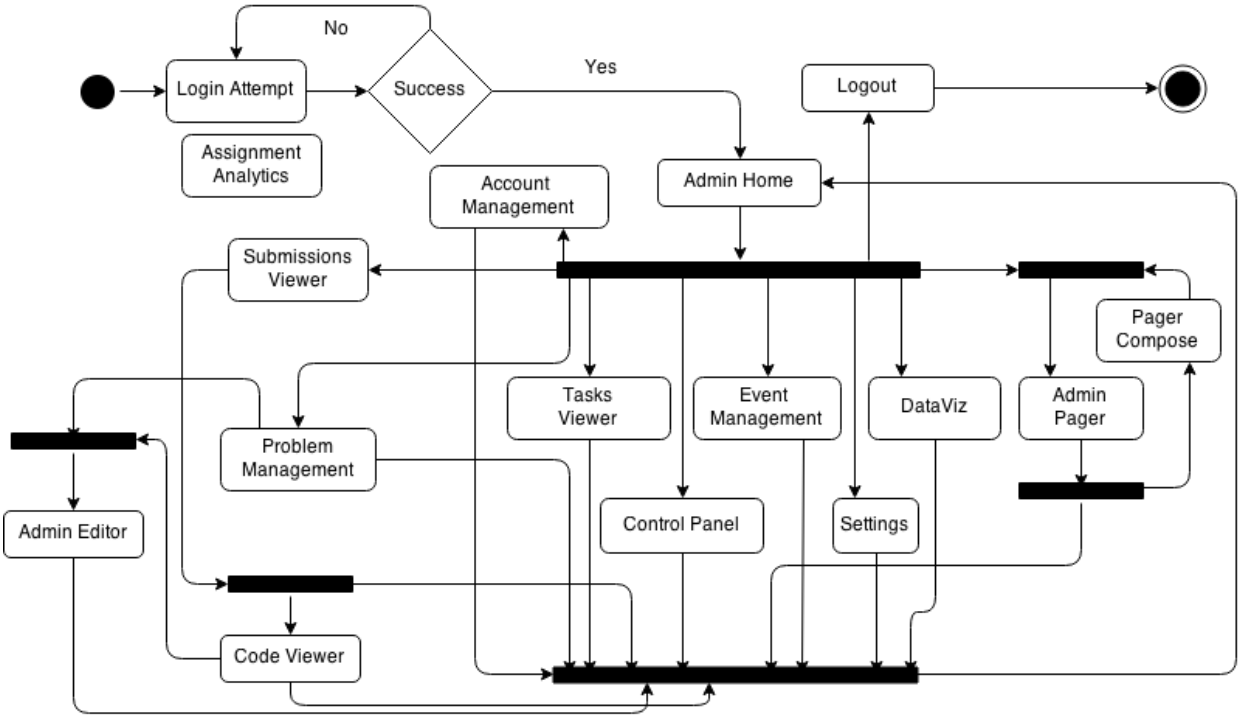

Figure 14: Normal admin activity

Development View

Web Application:

- All the application modules for the web application correspond to JavaScript files inside the “app_modules” folder of the application code base. The modules are in Common JS format. All external libraries required by the application modules reside inside of the “node_modules” folder. They are included in the respective application modules using Common JS “require” methods.
- All route controllers are methods that defined in their respective route files. These methods are registered with the application router which is a program within the Express JS framework. The route files are JavaScript files defined inside the “routes” folder of the application code base. These route files follow a hierarchy corresponding to the paths or routes that they handle.
- The view definitions are present inside the “views” folder of the code base. They are Jade template files that are used to render HTML views for the user interfaces.
- All static libraries are present inside the “public/vendor” folder. Their locations are registered in the “public/scripts/main.js” file, which is loaded in every user interface.
- The user interfaces are rendered using the view definitions along with JavaScript files and CSS files located inside the “public/scripts” and “public/styles” directory, respectively.

Engine:

- The bootstrap manager corresponds to the “core/bootstrap.php” file in the code base.
- The router functionalities are managed jointly by the “index.php” and “core/resolver.php” files in the code base.
- The registry is maintained by the class defined in “system/Registry.php” file.
- The session manager corresponds to the class defined in “system/Session.php” file.
- The classloader is defined for various types of classes in the “core/autoload.php” file.
- The controllers are defined in “app/controllers” as PHP class files. The routes to the methods in these controllers are defined in “app/config/ports.ini”. This file contains all the mappings from the request path to the controller and method which is to be invoked upon receiving the corresponding request.
- The application modules are defined in the “app/modules” folder as different PHP class files. These module classes are dynamically loaded by the classloader, whenever they are required by the application.

Dynamic Load Balancer:

- A single file (“updater.php”) manages the server list used by the dynamic load balancer. This file runs in the background as a daemon, and watches for updates to the server list used for load balancing. It updates the list and restarts the server whenever there is a change.

Auto Scale Monitor:

- A single file (“monitor.sh”) monitors the response times across the web servers (engine and web application). It looks for heat up and cooldown situations and does the necessary upscaling and downscaling respectively.
- The “upscale.php” file inside the respective service folder, is responsible for upscaling the web services corresponding to it. It spawns a new node and configures it for use by the system.
- The “downscale.php” file inside the respective services folder is responsible for downscaling the web services corresponding to it. It destroys a node from the system, when invoked.

Physical View

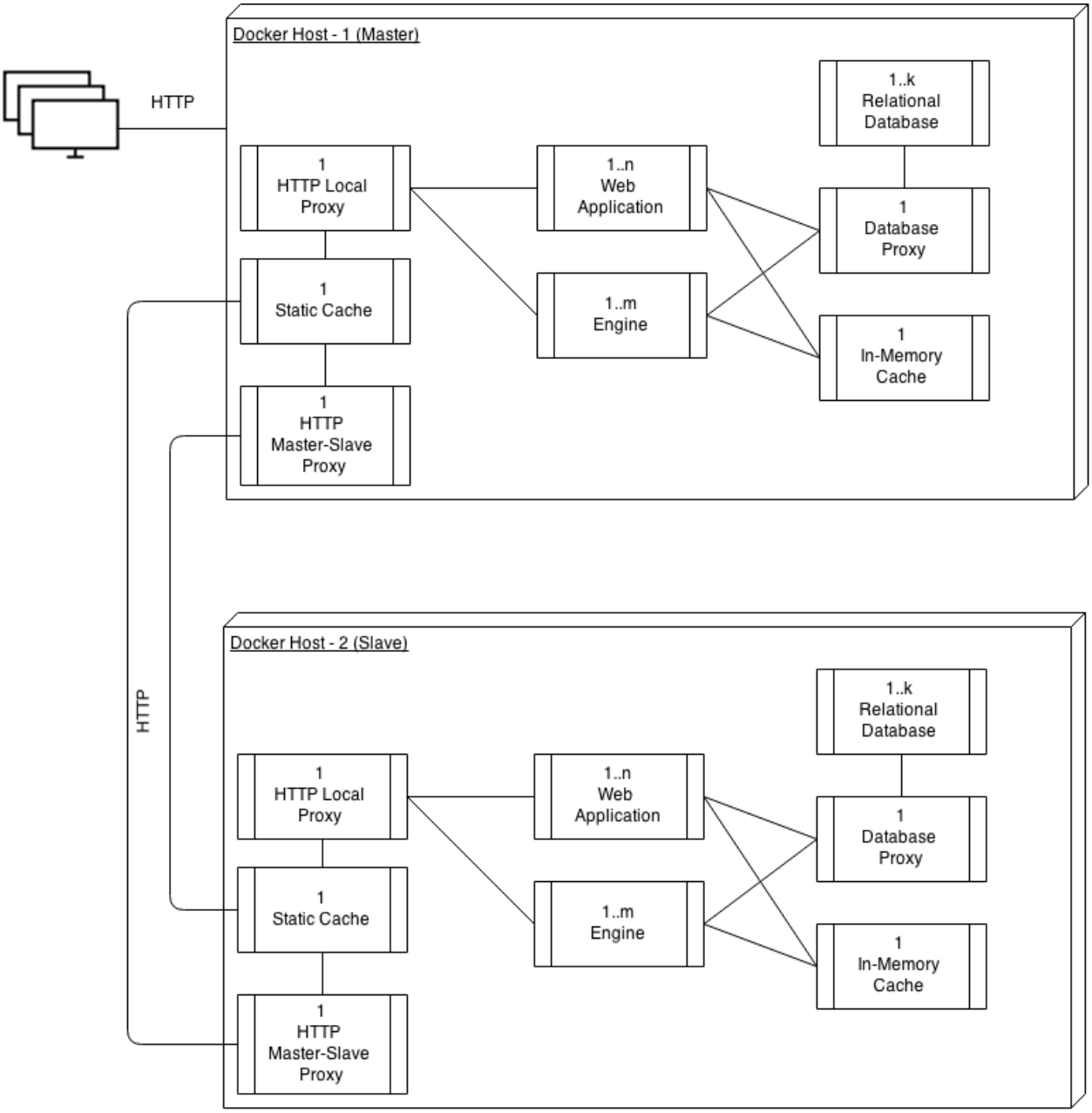

Figure 15: Deployment view of the tutoring system

CONCEPTUAL FRAMEWORK

Domain Lexicon

Engine: The component of the system which is responsible for tasks such as compilation, execution, evaluation and feedback generation.

Virtual File: An identifier for a student program, created on the system, which appears as a file to the student. Actual files are not created on the disk.

Virtual Folder: An identifier for a collection of programs, created on the system, which appears as a folder to the student. Actual folders are not created on the disk.

Workspace: A collection of virtual files and folders, used by a student for programming purposes.

ScratchPad: A web based IDE interface, wherein a student can create virtual files and folders as a part of her workspace. Students can arbitrary write programs in such files using a provided editor window, in the programming language assigned for the course. These programs can be compiled and their results can be viewed in the form of annotations in the gutter area of the editor window. Compiler messages can be viewed in a virtual console located within the same user interface. Further, the student may execute her compiled program on arbitrary test cases and view the results in an output window, within the same user interface. Execution results such as runtime errors or time limit exceeded errors can also be viewed.

CodeBook: A portal where students can view submitted solutions to programming assignments along with the corresponding problem statements for course events that were held previously. Grading information for the submitted assignments can also be viewed in the same interface. The portal also contains solutions to practice problems attempted by the student, along with the corresponding problem statements.

Course Event: Any event which is conducted during the course offering, and includes programming assignments to be solved by students. Such events are expected to have a set of programming problems assigned to each student who is enrolled for the course. The events may span multiple days or have multiple schedules. Lab assignments, examinations and quizzes are examples.

Schedule: A time period which corresponds to a course event, during which, members of that schedule may solve the programming problems assigned to them. All programming problems assigned to a particular student for a course event are only accessible to her during the time span of the schedule.

Practice Problems: A set of problems, which are not associated with any course event. They are accessible at all times, and can be solved by the student at all times. They are intended to provide practice to the students in solving programming problems.

PracticeArena: A portal where a student can access a collection of programming problems for practice. Students may view these problems and navigate to the editor interface, using which they can solve the problem.

Admin: A teaching assistant, tutor or an instructor of the course.

Teaching Assistant: A student who helps the instructor of the course in carrying out activities such as grading, invigilation and helping students solve programming problems.

Tutor: A student or a professor who has the responsibilities of setting questions for various course events and deciding the policies of the course events.

Instructor: A professor who conducts the course. He is responsible for deciding the course structure and grading policies for the course.

GradeCard: A table containing scores awarded by teaching assistants, tutors and instructors for the programming problems that were solved by the student for various course events. These scores are grouped by the course events attended by the student.

Pager: A publish-subscribe based messaging system wherein a student can create a message thread when she requires help in solving programming problems or in addressing technical difficulties, while solving the problems. Instructors, tutors and teaching assistants are able to view these messages and respond to them in real time. Only students can start a message thread, while others can only respond to them.

DataViz: A portal where data collected using the system can be tabulated in arbitrary formats or visualizations may be created from the data.

ControlPanel: A portal from where administrators can tune the settings of the Engine. Settings include time delays in compilation and execution along with execution quotas, compiler flags and tool active state.

CodeViewer: An user interface where the code history corresponding to a student submission for a programming assignment can be viewed. Information regarding the category of a code save along with any results of compilation or execution can be viewed for the entire code history. Code saves may also be deleted from the system using this interface. Further, admins can grade the programming assignments using this interface.

Lexicon Diagram

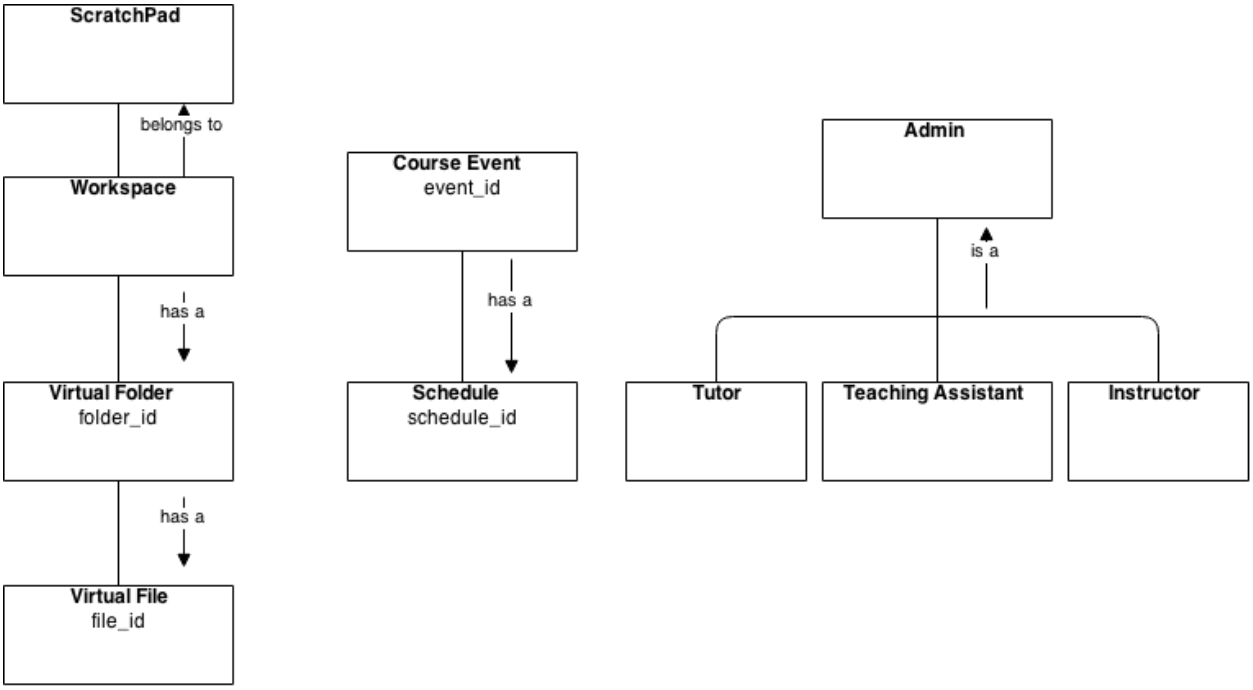

Figure 16: Lexicon diagram for concepts used

RESULTS

The following tables contain some results on scalability tests carried out for the system. Siege[37] was used as the stress testing tool for carrying out the tests.

Database write: Measures the write performance of the database. Hosts correspond to the physical machines being used. Two such machines were used in total.

Concurrency	Time	Hits	Host - 1	Host - 2
100	5 seconds	88	YES	YES
50	5 seconds	93	YES	YES
100	5 seconds	110	YES	NO

Compilation: Measures the performance of the system with respect to compilation requests.

Concurrency	Time	Transactions / Sec	Transactions (total)
20	5 seconds	92	452
100	5 seconds	94	425
400	5 seconds	91	392

Execution: Measures the performance of the system with respect to execution requests.

Concurrency	Time	Transactions / Sec	Transactions (total)
20	5 seconds	7	33
100	5 seconds	7	32
400	5 seconds	6	30

Evaluation: Measures the performance of the system with respect to evaluation requests. The compute expense column denotes the type of algorithm/program used for evaluation tests. The expense implies computational expense of the respective program. A high computational expense would imply that a lot of CPU time is utilized in each run of the program and vice-versa.

Concurrency	Time	Transactions / Sec	Transactions (total)	Compute Expense
20	30 seconds	1	20	HIGH

[37] Joe Dog, “Siege,” n.d., https://www.joedog.org/siege-home/.

100	30 seconds	0	0	HIGH
100	60 seconds	8	463	HIGH
100	30 seconds	8	215	MEDIUM
100	30 seconds	8	234	LOW

Web Application Nodes vs. Transactions: Measures the scalability of web application nodes across the two physical host machines. Nodes denote the Docker containers used and hence scalability implies scaling the CPU for the Docker containers. Multiple physical hosts are aggregated using a load balancer.

Host - 1 Nodes	Host - 2 Nodes	Transactions / Sec	Transactions (total)
2	0	29	141
3	0	42	211
4	0	59	283
5	0	73	353
5	1	78	320

Engine Nodes vs. Transactions: Measures the scalability of Engine nodes for a single physical host machine. Similar to the web application nodes, each Engine node corresponds to a Docker container within the physical host.

Nodes	Transactions / Sec	Transactions (total)
1	98	397
2	112	491
3	113	500

HTML Rendering: Measures the performance of the web application with respect to the HTML rendering engines. A lower response time implies a faster rendering engine.

Rendering Engine	Time	Responses
Jade	20 seconds	1963
DoT	20 seconds	33675

Student Home Page Loading Time: Measures the response time of the system while loading the student home page, which contains statistical information. This operation involves database

read accesses from the perspective of the web application. The response times indicate how well the system is able to handle a large number of requests.

Concurrency	Number of Requests	Response Time
20	1000	1500 ms
50	1000	4300 ms
100	1000	8400 ms
10	100	840 ms
5	100	430 ms

CONCLUSION

This system is architected in such a way that it can handle a load of about 400 students accessing the system simultaneously. It is easily deployable, owing to the use of Docker containers. This enables it to be deployed across multiple machines with ease and not requiring much expertise. Thus instructors of various programming courses may install this system on the servers available at their institutions. This system is also designed to run off any standard machine. This implies that it can even be run off a laptop machine. Most of the features provided by the architecture of this system reduce the amount of work required by system administrators in maintaining it. The highly modular nature of the codebase, provides a great framework for creating variants of this tutoring system.

The system has a combination of a layered architecture, a broker based architecture and a Model View Controller (MVC) based architecture.

Assessments

Docker is used as the virtualization environment for the system. This results in the system being easily portable. However, Docker's networking does not currently support multiple hosts. To address this issue, Weave has been used to create a common Docker network for containers residing on multiple hosts. Currently there are no issues to this approach, except for the fact that Docker itself does not recognize Weave. There is no way to obtain the information about what IP address is assigned to a container using the Weave network, via Docker. This has to be done manually by entering the container.

Multiple analytics tools have been integrated into the system, for research purposes. This has proved the system to be extensible. These tools were integrated by simply adding a new module into the Engine subsystem. The web services provided by this module were then registered in the routes of the Engine.

The system had been modified to work with the Python interpreter as well as some basic Prolog. The required changes to the system took around 2 days with a single person's effort. In general, in order to allow the system to work for a new imperative programming language, the only modification required is to change the configuration files. The system can also be moulded to include other tools as well, such as feedback tools. This makes the system highly modifiable.

The auto scaling algorithm had been tested by subjecting the system to stress tests. The system was flooded with requests gradually, and the number of nodes increased to a specific value. Similarly, when the flooding was over, the newly spawned nodes were destroyed after some time. Scalability tests were also carried out manually. The response times (a measure of the performance of the system), decreased for the web application and engine subsystems, as more nodes were added to the system. According to the benchmarks, this was somewhat linear. The database also scaled slowly up to a certain number of nodes, till it reached the saturation

point. This was understandable, as the clustering scheme used for the database was synchronous replication. In order to scale up the database further, the data model of the system needs to be changed.

There has been no data loss in the system till now. Failure of nodes did not affect the integrity or consistency of data in the system. Removal of database nodes from the system did not result in loss of data. This is consistent with the fact that synchronous replication is being used. Thus the system is indeed durable.

There had been several occasions when a number of nodes of the system had crashed, due to bugs in software or other reasons. However, service was not disrupted and the failures were invisible to most of the end users. Only the few, who were using the respective node at the time when it crashed, were affected for a brief period of time. This is due to the fault-tolerant design of the system, where multiple redundant nodes are kept to mitigate failures.

FUTURE WORK

This system offers a wide range of features to cover most of the basic requirements in conducting a programming course. Yet, there is a lot of future work, which needs to be done in this area. To begin with, it must support a large selection of programming languages and paradigms, so that instructors may be able to choose the language appropriate for their course. The tools which have been designed for feedback generation must be able to provide feedback for all programs, irrespective of the language in which it is written. The platform should support seamless integration of any sort of feedback tools without the aid of any developers. In addition to feedback generation tools, the platform should support the integration of problem generation tools, once problem generation technology in the field of programming is stable. Similar support should be present for test case generation, where KLEE[38] is currently being used. The system should be able to automatically generate problems and their test cases, when provided with specifications for the same. There should also be interfaces to integrate automatic grading tools into the system. Currently teaching assistants or instructors need to manually grade the student submissions to programming problems. The system should export additional statistics to reveal the progress of the student in learning the course concepts. Further, it should log data which may help to understand the deficiencies of the students in the course concepts.

Currently, the system presents the same perspective to all students enrolled for the course. In order to improve the effectiveness in learning, one needs to realize the fact, that different students may have different learning speeds and aptitude. Further, one student may be weak at concepts different from another student. The system must address this issue and provide different perspectives to different students based on their performance and coding history. Different problems with different levels of difficulties must be generated for different students. The feedback provided to the students must also differ among each other.

From an architectural perspective, this system is currently designed to be deployed in house for courses conducted in a university. One may want to reach out to a large number of students across the world, in a way similar to MOOCs. Although this can be done currently, the architecture is not efficient for this purpose. This is in fact due to the data model of the system, which needs to be revised and remodelled for scalability. Designing a non-relational data model will permit the use of NoSQL[39] database systems such as Cassandra[40] and MongoDB[41]. Such database systems can scale to a huge number of users and will permit the system to be used for MOOCs.

---

[38] Cristian Cadar, Daniel Dunbar, and Dawson Engler, “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs,” in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI’08 (Berkeley, CA, USA: USENIX Association, 2008), 209–24, http://dl.acm.org/citation.cfm?id=1855741.1855756.
[39] “NoSQL,” *Wikipedia*, n.d., http://en.wikipedia.org/wiki/NoSQL.
[40] “Cassandra,” n.d., http://cassandra.apache.org/.
[41] “MongoDB,” n.d., https://www.mongodb.org/.

BIBLIOGRAPHY

Cadar, Cristian, Daniel Dunbar, and Dawson Engler. “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs.” In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, 209–24. OSDI’08. Berkeley, CA, USA: USENIX Association, 2008. http://dl.acm.org/citation.cfm?id=1855741.1855756.

“Cassandra,” n.d. http://cassandra.apache.org/.

Joe Dog. “Siege,” n.d. https://www.joedog.org/siege-home/.

“MongoDB,” n.d. https://www.mongodb.org/.

“NoSQL.” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/NoSQL.

Ahmed, Umair Z., Sumit Gulwani, and Amey Karkare. “Automatically Generating Problems and Solutions for Natural Deduction.” In *IJCAI 2013*, 2013.

Alur, Rajeev, Loris D’Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. “Automated Grading of DFA Constructions.” In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 1976–82. IJCAI ’13. Beijing, China: AAAI Press, 2013. http://dl.acm.org/citation.cfm?id=2540128.2540412.

Apache Software Foundation. “Apache HTTP Server,” n.d. http://httpd.apache.org/.

“Automata Tutor,” n.d. http://www.automatatutor.com/.

Bakuya, T., and M. Matsui. *Relational Database Management System*. Google Patents, 1997. https://www.google.com/patents/US5680614.

Cardellini, Valeria, Michele Colajanni, and Philip S. Yu. “Dynamic Load Balancing on Web-Server Systems.” *IEEE Internet Computing*, 1999.

Cingi, Can Cemal. “Computer Aided Education.” *Procedia - Social and Behavioral Sciences* 103 (November 2013): 220–29. doi:10.1016/j.sbspro.2013.10.329.

“Coursera.” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/Coursera.

“Coursera.org,” n.d. https://www.coursera.org/.

D’antoni, Loris, Dileep Kini, Rajeev Alur, Sumit Gulwani, Mahesh Viswanathan, and Björn Hartmann. “How Can Automatic Feedback Help Students Construct Automata?” *ACM Transactions on Computer-Human Interaction* 22, no. 2 (March 10, 2015): 1–24. doi:10.1145/2723163.

dotCloud. “Docker,” n.d. https://www.docker.com/whatisdocker/.

“Edx,” n.d. https://www.edx.org/.

Galera. “Galera,” n.d. http://galeracluster.com/products/technology/.

Gulwani, Sumit, Vijay Anand Korthikanti, and Ashish Tiwari. “Synthesizing Geometry Constructions.” In *PLDI*, 50–61, 2011.

HAProxy. “HAProxy,” n.d. http://www.haproxy.org/.

Hashicorp. “Consul,” n.d. https://www.consul.io/.

“Integrated Development Environment.” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/Integrated_development_environment.

Joyent. “Node JS,” n.d. https://nodejs.org/.

“Khan Academy,” n.d. https://www.khanacademy.org/.

Krasner, Glenn E, Stephen T Pope, and others. “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.” *Journal of Object Oriented Programming* 1, no. 3 (1988): 26–49.

“Load Balancing (computing).” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/Load_balancing_%28computing%29.

“Massive Open Online Course.” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/Massive_open_online_course.

Memcached. “Memcached,” n.d. http://memcached.org/.

Michael A. Ogush, Derek Coleman, Dorothea Beringer. “A Template for Documenting Software and Firmware Architectures,” March 15, 2000. http://www.cs.helsinki.fi/group/os3/HP_arch_template_vers13_withexamples.pdf.

“NPTEL,” n.d. http://nptel.ac.in/.

Oracle. “Java EE,” n.d. http://www.oracle.com/technetwork/java/javaee/overview/index.html.

———. “MySQL.” *MySQL*, n.d. https://www.mysql.com/.

PHP. “PHP,” n.d. http://php.net/.

Polozov, Oleksandr, Sumit Gulwani, and Sriram Rajamani. *Structure and Term Prediction for Mathematical Text*. Microsoft Research, 2012.

“Proxy Server.” *Wikipedia*, n.d. http://en.wikipedia.org/wiki/Proxy_server.

Redis. “Redis,” n.d. http://redis.io/.

Singh, Rishabh, Sumit Gulwani, and Armando Solar-Lezama. “Automated Feedback Generation for Introductory Programming Assignments.” *ACM SIGPLAN Notices* 48, no. 6 (June 23, 2013): 15. doi:10.1145/2499370.2462195.

Singh et al. *Automated Semantic Grading of Programs*, 2012.

Thomas Orozco. “Scalr Auto Scaling Algorithm,” n.d. https://scalr-wiki.atlassian.net/wiki/display/docs/Autoscaling+Algorithms.

“Udacity,” n.d. https://www.udacity.com/.