ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

AUTOMATED CODE AND CONTEST EVALUATION SERVICE (ACCES)

Graduation Project Final Report

Kadir Emre Oto
150140032
Muhammed Burak Buğrul
150140015
Süheyl Emre Karabela
150140109

Department: Computer Engineering
Division: Computer Engineering

Advisor: Asst. Prof. Ayşe Tosun

January 2020

Statement of Authenticity

We hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by our individual authenticity.

03.10.2019	03.10.2019	03.10.2019
Kadir Emre Oto	Muhammed Burak Buğrul	Süheyl Emre Karabela

AUTOMATED CODE AND CONTEST EVALUATION SERVICE

(SUMMARY)

Coding assignments are very popular not only in computer engineering departments but also in almost every engineering department. Classic process for the lecturers/T.A.s:

- Give the homework statement and a couple of examples.
- Collect students' codes by the deadline.
- Evaluate them with a test set and analyze them one by one (most time-consuming part).
- Announce the grades.

The 3rd step can take months for a crowded course. But it is generally a deterministic step. All the codes submitted by students should be tested with the same test cases. And every code should be analyzed in same aspects like memory usage, run time etc. An automated system can do this within seconds instead of months. In addition, with a fast system, students can have a feedback mechanism. This situation leads students to learn and practice more efficient and T.A.s to save their times.

ACCES targets these outcomes within a scalable, secure and robust system. In addition, it will be a Software as a Service solution for every school to use it easily.

In point of view of a student; when a code is submitted for a homework, process afterwards is unknown. Which type of test cases failed, which type of test cases passed, how was the performance of the code etc. Event he/she cannot know whether the code run or crashed in the assistants' computer. But in ACCES, run environment of the projects are known and fair for every student. A student can run his/her code before submitting the homework to analyze its' behavior. After submission, he/she can see which test cases passed and failed. In addition, he/she can see the data of test cases if assistant or lecturer allows. At the end, he/she can see an expected grade of the submitted code. Of course, it may not be the final grade.

In point of assistants, they can be assigned to courses and manage students and homework of assigned courses. They can give homework and determine test data for the homework. They can split the test data sample (visible to students) and private (only the pass/fail status visible to students) sets. They can add or delete test cases whenever they want. All they need is a working solution for the homework, addition/deletion of parts can be done via ACCES website. When they prepare everything for a homework, they can evaluate submitted codes with one click in minutes instead of months. In classical evaluation processes, changing the test data leads T.A. to reevaluate all of the submitted codes. In ACCES, re-evaluation is a one click operation as well.

In point of view of lecturers, they can do everything assistants can do, in addition they can manage assistants.
Besides the benefits from the view point of users, ACCES ensures some technical specifications:

- Security: Every user communicates with the system by using time-based authentication tokens and every code run operation handled by the judge is in a separate sandbox.
- Performance: We use the back-end in a functional 'do it and disappear' method in order to shorten the data travel path between users and the system. In addition, with this methodology, every endpoint can be scaled in different machines. This scale option provides handling high request/seconds operations easily.
- Availability: System saves the data in more than one place. And our databases are different services. Other than the database records, we log everything in a separate service.

In conclusion, our motivation to encourage usage of ACCES in schools is providing almost equal opportunities to students, making the code evaluation process as transparent as it can be, making assistant's job easier and establish a good feedback mechanism.

Contents

1. Introduction and Problem Definition

Most universities, especially in Turkey, do not have a fully automated homework or exam evaluation system. An automated system can reduce the evaluation time hundreds of times while increasing accuracy for code assignments in computer engineering and similar departments. For this reason, instructors have to evaluate the student submissions manually (sometimes this process takes 2 or 3 months), and students are not able to get any feedback about their submissions until the grades are announced months later.

In this project we are offering a system that can evaluate the student's submissions within minutes and give the necessary feedback to students about their submission. In this way, we are planning to use this system in coding exams and other similar cases that take lots of time to evaluate. For that matter:

- Students can see their grades and code errors in a short while.
- Students will have the opportunity to correct their codes by receiving quick feedback (at most 5 minutes instead of 3 months).
- Instructors won't waste their precious time doing these chores.
- The probability of human error will reduce to almost zero.

2. Literature Survey

In this section, both competitive programming platforms and educational platforms will be addressed.

HackerRank:

HackerRank is a platform that enables companies to determine the skills of developers by providing programming challenges and online environments [1]. Other than business solutions, users can also use the site to solve programming challenges provided by HackerRank in many domains to improve their coding skills and attend to the contests that are organized by them [2]. HackerRank uses a cloud-based code execution and evaluation system which has a similar function and operating method to our code evaluation system. It uses the system to execute the user's codes submitted to the challenges with the predetermined inputs to check if they give the correct outputs based on custom grading functions or exact match to the predetermined outputs. The memory and time constraints for executions can be configured. Our judge system will have similar functionality, it will support the same input and custom grading-based evaluation methods and configurable time and memory constraints. The main difference between the HackerRank and ACCES is that ACCES provides automated homework evaluation system for academic institutions while HackerRank helps companies hire skilled developers.

Codeforces:

Codeforces is an online competitive programming and social network platform [3]. It has a large number of challenges in various difficulties. It holds many rated and unrated contests which the rated ones affect the rank of the participant. It is more contest oriented than HackerRank and do not provide hiring products to businesses. It has a similar online judging system to HackerRank and ACCES.

CSAcademy:

It is a competitive programming website [4]. It contains some problems that require a one file batch solution. In addition, it holds some online contests for competitive programming enthusiasts. The most important aspect of CSAcademy that differs it from other competitive programming platforms is its applications. There are 3 applications in the platform:

- Graph Editor: It allows one to create directed/undirected, weighted/unweighted graphs and change places of nodes. Through this editor, the test inputs can be visualized dynamically.
- Geometry Widget: It allows one to create and replace simple geometric shapes in a 2-dimensional plane.
- Diff Tool: It accepts two texts as input and returns a detailed difference output of these two input texts. Details are like in the vimdiff tool in unix systems with colors. This tool can also be used to highlight the differences in code pieces.

Calico:

"Calico is a utility for checking command-line programs in terms of their input and output". It checks whether a program creates the right yield when given a few sources of info. It was created to assess straightforward programming assignments in an introductory programming course. [5]

Instructors of BLG102E Int to Scientific & Eng.Computing (C) at ITU are currently using this python module. But it only provides the exact input-output checking via command line.

GradeScope:

It is an automated exam evaluation service [6]. It works on predetermined, partial point milestones and gives notes to students according to them. It can detect handwritings, and it can be used for an online objection system.

3. Developed Approach and System Models

3.1. Definitions

System: A system that automates code evaluation process for computer science related courses.

User: An authenticated person:

- **Student:** Can view enrolled course materials and can submit homework and see result of homework via ACCES.
- **Teaching Assistant:** Can organize materials of related courses according to given permission by course owner (Lecturer)
- **Lecturer:** Can create courses, add enrolled students and assign assistants to his/her courses. Also can give permissions on course materials to assigned T.A.s.
- **School Admin:** Can organize lecturer/assistant/student registrations to the system.
- **System Admin:** Can create and assign new systems to new schools. Can monitor system situation and interfere into the system if required.

Profile Page: Shows enrolled courses, announcements, user info etc. One can navigate to lectures and related announcement pages from profile page.

Profile Settings: A user can change his/her password etc.

School Selection Page: A user can select one of his/her current schools.

Lecture Page: Shows course info (Lecturer, time, grading etc.) and homework. One can navigate to related pages form lecture page. Also provides course adding features.

T.A. Page: Shows lecture info (Lecturer, time, grading etc.) and homework. One can navigate to related pages form lecture page.

Homework Page: Shows homework info(statement, deadline, language constraints). Includes an online editor. One can write and submit his/her homework code via this editor. Afterwards he/she can see submission results.

Online Editor: Organizes a file system for homework. Students can change permitted fields on this editor. Also this editor provides code coloring.

Course Message Panel: Shows message threads created by students, T.A.s and lecturers.

Course Message Thread: One single topic page, shows all messages sent to related topic in chronological order. Also, one can submit his/her own message to related thread.

Course Settings: Lecturer can change permissions of T.A.s and add/update/delete homework.

Course Adding: One with the permission can add course with materials.

Homework Adding: One with the permission can add homework with materials (statement, sample inputs/outputs, language constraints, judge type etc.).

Homework Settings: One with the permission can change settings.

3.2. Project Resources

- Software
    - PyCharm
    - VSCode
    - Termius
    - Adobe XD
    - Postman
    - Docker / Kubernetes
    - Frameworks
        - Facebook React
        - Google Firebase
        - Google Pub/Sub
        - Palantir Blueprint
    - Judge Daemon Resources
        - G++ 14 compiler
        - Python 2.7
        - Python 3.7
        - Java 8
        - Java 11
        - IOI Isolate
        - Docker
- Hardware
    - 3 servers for judging (Digital Ocean)
    - 3 laptops for developing

3.3. High Level Design

3.3.1. Back-end

Back-end of ACCES, consists of several different purposed systems. We avoid monolithic approaches to develop and manage project parts separated and handle scale operations easily and robust.

Static File Storage: All static files(Task statements, input/output files of tasks etc.) are stored in this service.

NoSQL Database (Cloud Firestore): It is a JSON document based NoSQL database. All documents resides in a named collection. Any document can include collections.

Database Triggers: This service listens the database and takes actions according to the changes. For example it grades a task submission of a student when the judging is done.

Judge Daemons: Judge machines. They are the machines that runs the tasks.

Message Queue: It distributes tasks among Judge Daemons. It supports publish and subscribe actions.

Logging: All system action and logs goes here. This system is vital when it comes to error tracing.

Stateless and Scalable Functions: These are back-end endpoints. They are independent from any other service. This independence provides a functional programming like environment for the functions. They become stateless and easily scalable. Scalability is crucial when there are thousands of concurrent users.

Google Firebase Hosting: This service distributes the front-end application to clients.

Client: Any user that uses the ACCES website.

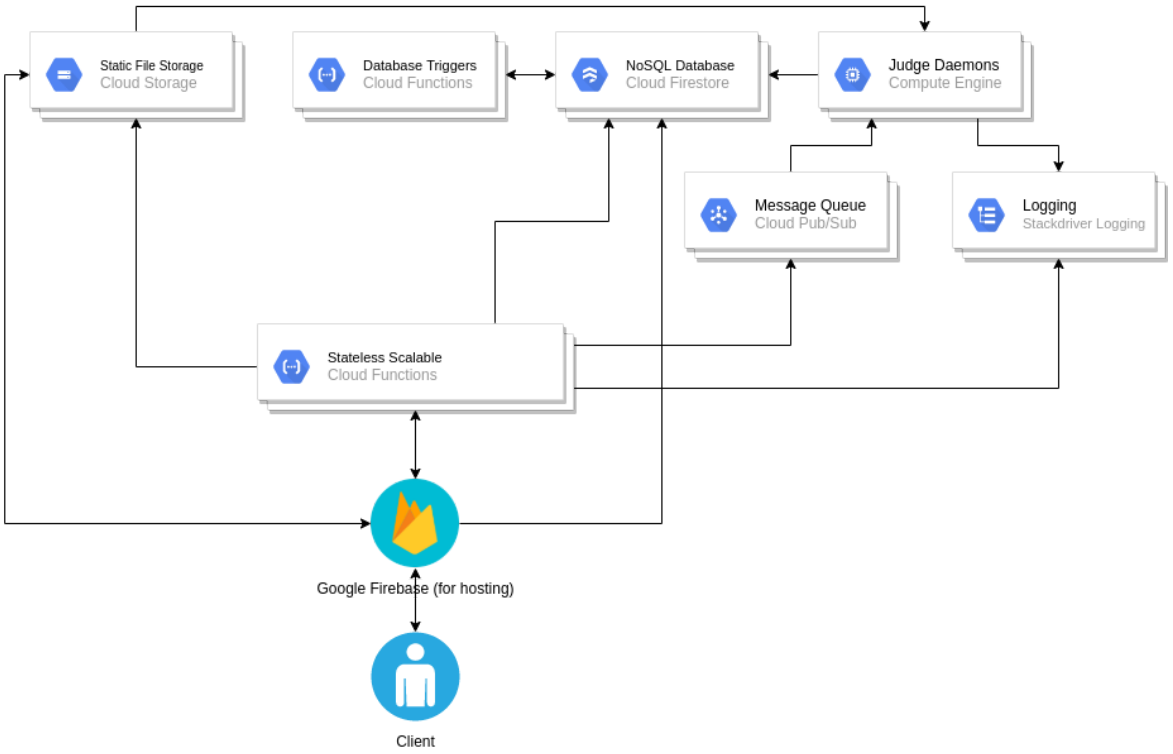

Figure 1: High Level Architecture of ACCES [7]

3.3.2. Evaluation System (Judge)

Purpose of a Judge daemon is to get tasks to evaluate from the message queue. When a student submits a solution for a task, it will be sent to the message queue after back-end checks and adjustments. A daemon listens to the message queue and takes submissions according to its' computing power. When the evaluation is done, it writes the results to the storage and the database. All of the daemons work independently. So, any number of daemons can be added to the system easily.

DBManager: It is the connection between the database (Google Cloud Firestore) and the system.

FileManager: It is the connection between the file storage (Google Cloud Storage) and the system.

Consumer: It listens the message queue (Google Pub/Sub) and get information about task that will be evaluated within the system.

Submission: Submission model class. It holds the information related to a submission like submit time, task information, assessment information etc.

Judge: Judge library communicates with the isolated code running environment and runs the submission and check whether its' output are correct.

Result: Result model class. It holds the information related to a submission result like correctness, run time, memory usage etc.

Daemon: It manages the judge services.

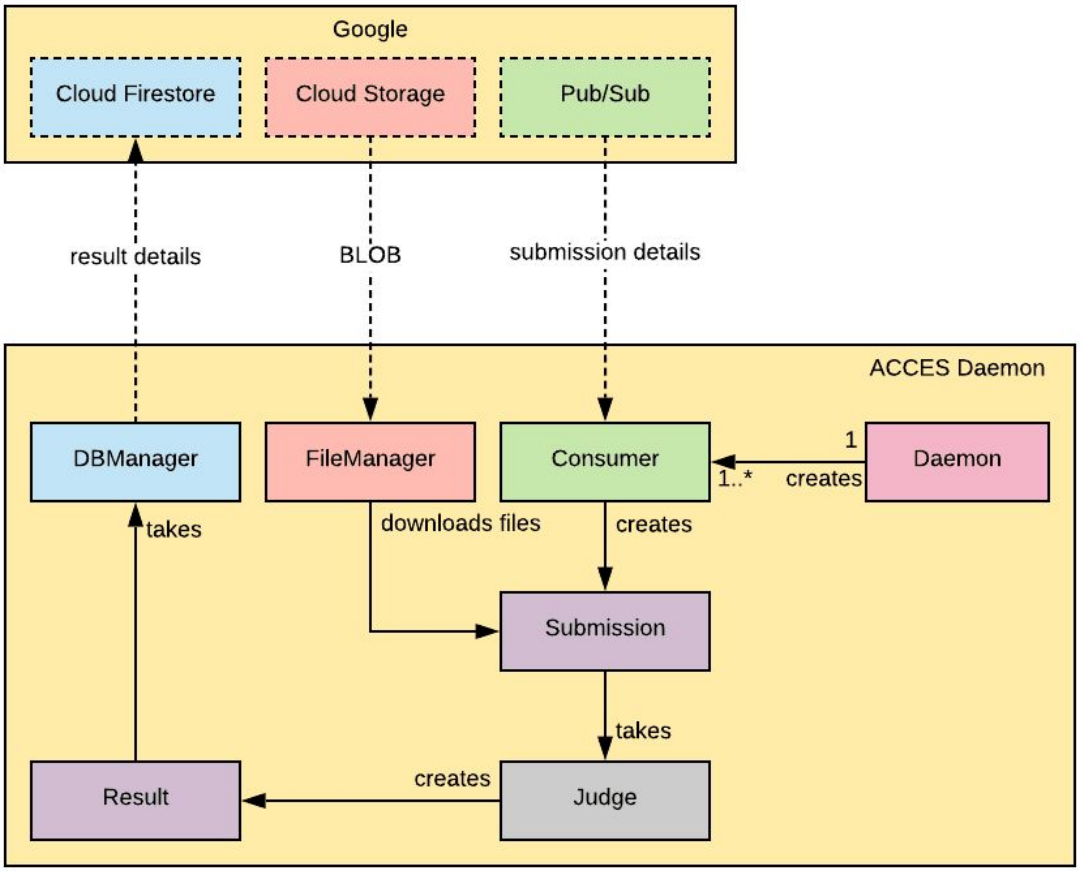

Figure 2: High Level Architecture of Judge Microservice[7]

3.3.3. Database

Main database of the system is Google Firestore. It is a document-based NoSQL database.

We choose a NoSQL database because our database design is flat and it should be held in different shards. One other advantage of Google Firestore is that it provides live database listener to detect changes on specific documents or collections as fast as possible. For performance boosts, there are data repetition and there are no explicit relations between database entities. These choices lead us to choose Google Firestore as our main database. ACCES database entities are:

users: A collection of user information documents.

organizations: A collection of organizations like schools, companies etc.

courses: A collection of courses and their information like lecturer, assistants etc.

assessments: A collection of assessment documents. An assessment can be a homework, midterm, lab. assignment etc.

tasks: A collection of tasks. Tasks can be attached into assessments implicitly (by adding their IDs into the assessment). A midterm task can be a question, a homework task can be a coding practice etc.

submissions: A collection of submission information. When a student submits code(s) to solve a task, a submission document will be created. It hold related information (code paths, input/output results etc.). After judging, results will be written in the same object as well. In addition, for every output of the task, there is a separate document in submission document. This enables front-end service to listen to these result documents and show the judging result to the user live.

public_grades: A collection of assessment-student result pairs shown to students.

full_grades: A collection of assessment-student result pairs shown to lecturers and assistants.

runs: A collection of students' current states of the tasks.

daemons: A collection of current judge daemons and their statuses.

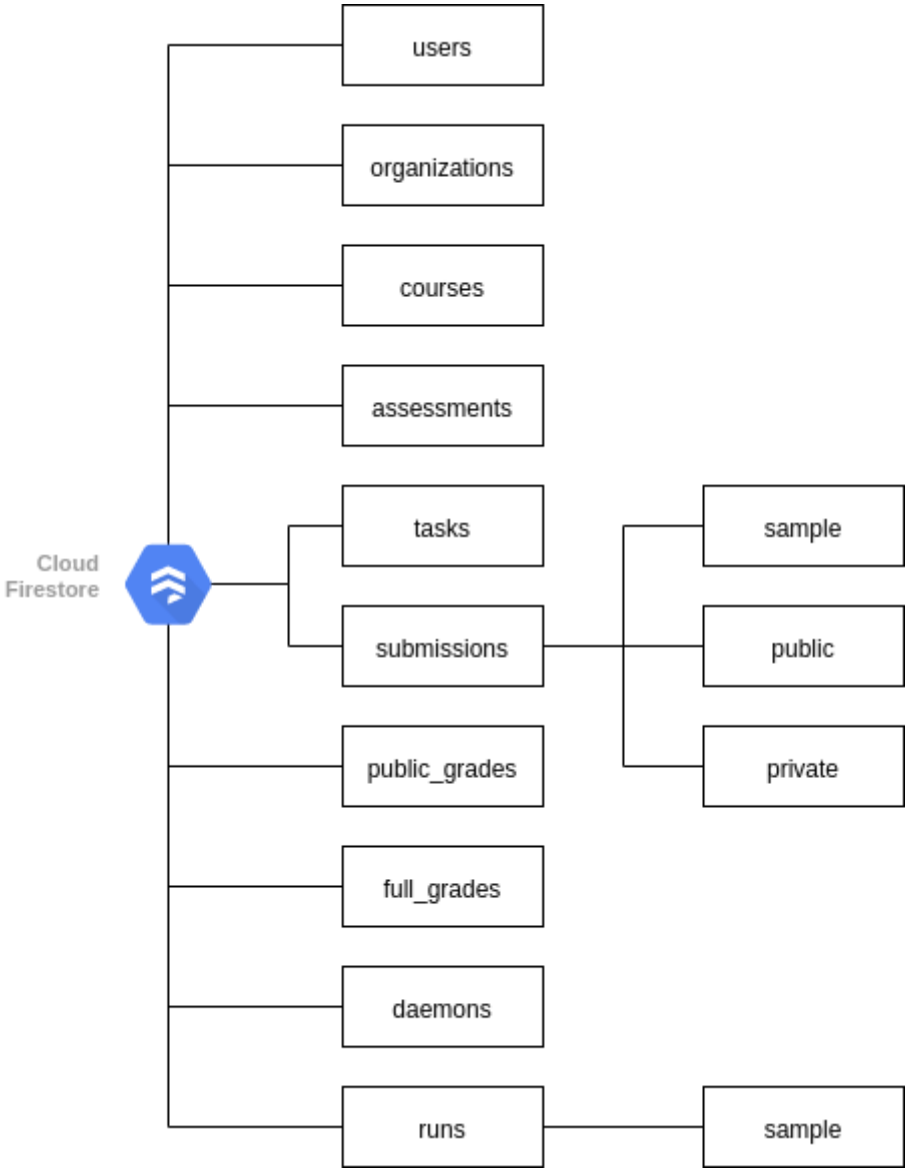

Figure 3: Database entities [7]

3.3.4. Online Code Editor

One of the main features of ACCES is the online code editor platform. ACCES is not only an assessment organizer and grader system but it is also an easy to use online coding platform. In addition, it behaves like a workspace by saving projects last statuses (last changes on the codes). It provides a user interface that includes information windows and implements two different actions: 'Run task' and 'Submit task'.

Run task: This action submits the student codes to run it with 'sample test cases' in judge daemons and shows output, expected output and error output. Sample test cases are test cases that students can show all information about them.

Submit task: This action submits the student codes to run it with all of the test cases. It is an official assessment submission. After submission, the student can see the results of all of the test cases at the bottom of the editor once they are evaluated.

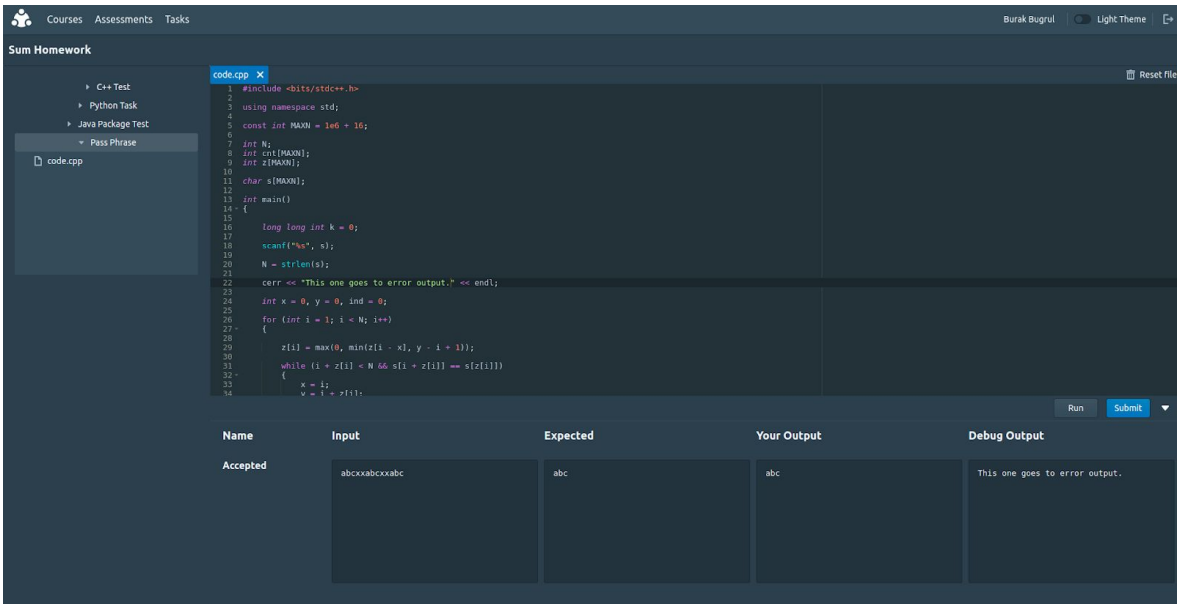

Figure 4: Run Action

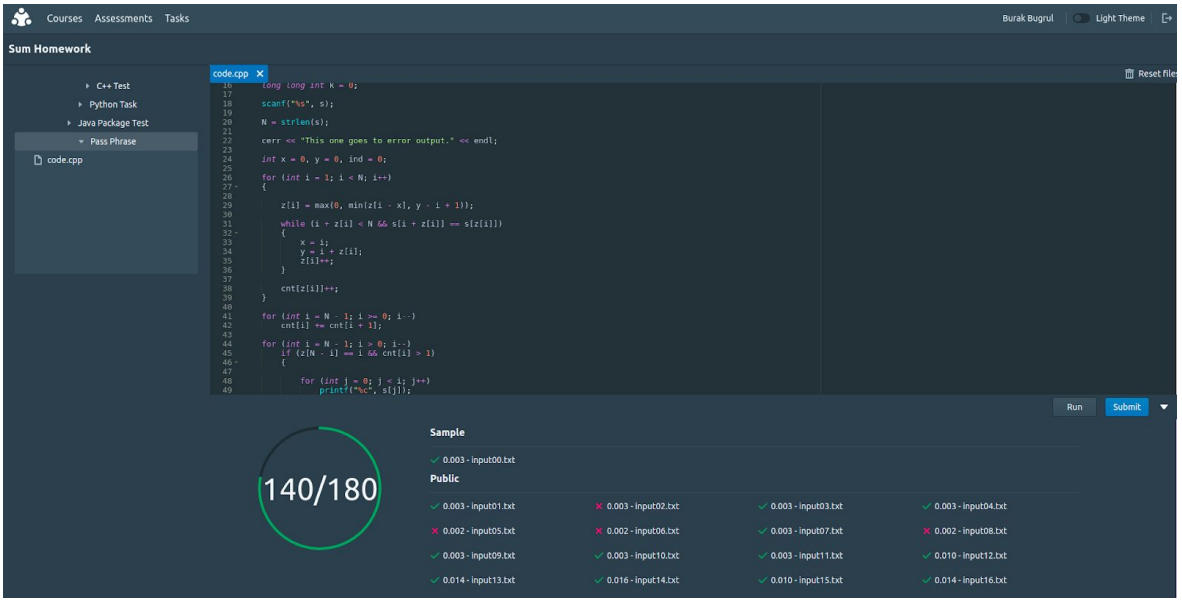

Figure 5: Submit Action

Interfaces in the editor helps students to organize and develop their tasks.

Project Structure: On the left part, the student can navigate among tasks and their files. Whenever he/she clicks a file, the content of that file appears on the code screen.

Code Screen: It is the part that shows the content of the files and allows students to edit them. It provides a simple code completion feature.

Run Action Results: When a task run against sample test cases, a result component appears at the bottom of the editor and shows related output and error information.

Submit Action Results: When a task run against all of the test cases, a result component appears at the bottom of the editor again and show the result of every public test case to the student. Each test case results have hovering information boxes for giving details about the result (run time, result and test case name). Earned score from the submission is shown on the left, when everyone of the test cases are evaluated.

4. Experimentation

There are several commitments when it comes to performance, features, security and availability. We performed various tests on these aspects.

4.1. Performance

It can be examined from different points of view. Number of different users supported, scalability of back-end endpoints, throughput of judge daemons when their CPU's are nearly at full computing state etc.

Number of different users: We use Firebase authentication service at highest subscription plan (Blaze plan). There is no physical limit about our own servers at this point.

Back-end Scalability: We use stateless functions as back-end endpoint. They are independent from database, storage and data actors. They are fully deterministic and CPU dependent. So, all of the endpoints can be scale horizontally among any number of computing machines. We use Serverless Google Cloud Functions service to utilize the scale according to traffic. When the traffic is high, functions scale with other machines. When the traffic is low, functions will be downscaled. We continuously sent requests to the back-end functions batch by batch in order to analyze scale behavior.

Batch Number	Request per Second
1	350
2	1540
3	4060
4	12050

We tested with a function that does 3 database reads and 1 database write operations.

Throughput of Judge Daemons: Judge daemons always listen to the message queue and get student submissions to evaluate according to their computing power. We put different numbers of submissions in the message queue and measure their average consume time. Specs of the machine we used as the judge daemon in this test are:

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 6 physical 6 virtual CPU cores
- 16 GB Memory
- 20 Mbit stable internet connection

This judge daemon run 12 consumer workers and it is tested against 3 different submission scenarios.

Number of active test cases	Average evaluation time per test case
486	0.3 seconds
990	0.4 seconds
4992	0.6 seconds

12 workers can run parallel when judging the test cases. Independence of the test cases enables easiness of adding new judge daemons.

4.2. Security

Security issues should be at minimum when it comes to formal student/school relations. Unlike many other systems, we implemented security rules not only in back-end endpoints, but also in web application, storage service and database service. This security rules provides a simple check for both authentication and authorization in each one of the internal and external endpoints. Due to the fact that there are 4 major rule checkpoints that one can leak or corrupt data, all of them must be broken at the same time because their processes are different and independent. Besides these rules, there are upload and resource limitations for all users. No one can upload or download files limitlessly.

Judge daemon limits every submission within a sandbox environment. Whatever is done is done within the environment. In addition, there are run time, memory and output size constraints to protect resources as well.

4.3. Availability

Almost all of the microservices are open to horizontal scale operations. ACCES is consist of these microservices for not only scalability but also making DevOps part of the project easier and stable [8].

Databases have replicas within different regions. These replicas ensure fault recovery for the data. But there is a trade-off between making every update in every replica and fault recovery system [9]. In ACCES, this is balanced with the help of Google Firestore.

In the system, network traffic will not be uniformly distributed over time. So, there is no point of running all of the powerful application servers all the time. Number of

application servers should be scaled automatically according to the traffic [10]. With Google Cloud functions and Google's Kubernetes, scaling is done automatically.

These replications and scaling techniques enable high availability both for the services and the data. In addition, it provides a fault recovery system, whenever a processing node is down, other replicas can answer to the request and the broken node can be repaired in the background smoothly [11].

4.4. Quality Assurance and Development

Quality assurance and maintainability for a long term and big scaled project is crucial. Organization should be structured and neat even from the beginning of the project planning sessions.

In the planning part, every idea classified by priority and wrote in a Kanban board on Trello [12]. Kanban boards enables task assignment for every member of the project. By doing this, tasks and members are effectively followable [13]. Parts of the Kanban board in this project:

- **Intro**: Information about the board and ACCES.
- **Backlog**: Tasks to be discussed. They don't have high priority.
- **To Do**: Well defined tasks waiting for someone to pick them.
- **Urgent**: Urgent tasks to do.
- **Bugs**: Logs of known bugs. They are tasks as well.
- **In progress**: Tasks that picked by someone and in progress.
- **Testing**: Finished tasks, waiting for testing.
- **Done**: Successfully ended tasks.

In order to maintain codebase and develop features parallel, we used GitLab [14] as version control system. ACCES organization have 3 different repositories in GitLab: Judge Daemon, Back-end Functions and Web Application. For stability, we defined a commit structure as well: "[Past tense verb] [object] [other information], [other commit messages...]".

In the development cycle, a member is assigned at most 3 tasks at the same time. And a task followed To Do -> In Progress -> Testing -> Done path. We maintained all of the tasks as minimal as possible because we are working with microservices. At the end of the path, the related microservice should be updated. We integrated our development with continuous integration pipeline. All of the microservices are updated directly from codebase using graphical interface. And thanks to scalable nodes, no system is down when its' microservice is in updating state. We believe a smooth user experience is important as structuring the deployment operations. [15].

5. Evaluation

5.1. Evaluation Criteria

As mentioned in the requirements section, the system should be highly available, secure and robust. These are the numerical goals:

- The system should support more than 10.000 concurrent users.
- The system should not leak personal information of its users.
- User codes should be executed in an isolated environment in order not to crash the system.
- In case of a system failure, the system should be recovered in 30 seconds.
- The system web apps should be accessible via using at least LTS versions of Google Chrome, Mozilla Firefox, Microsoft Edge and Safari web browsers
- All submissions should be judged in 1 minute.
- An assistant can add students to a course.
- An assistant can add homework to a course.
- An assistant can modify test data of a homework.
- An assistant can evaluate homework submissions.

5.2. Evaluation Results

Possible security, availability issues measured, precautions are implemented and designed as above. For the performance, the system tests (scalability and judge tests above) satisfied the criteria. Features are subjected to acceptance tests according to the use case scenarios in requirements part of the project planning phase and system is working as it should. It is ready to use for a school for the moment. The key points is, a judge daemon can grade an assignment submission of a student in seconds while an assistant can do the same in at least minutes.

6. Conclusion and Future Work

Problem: Most university departments started to assign coding problems in student homework and exams, but the evaluation and grading process takes a long time such that students may need to wait more than four weeks to see their grades, and academicians have to spend a considerable part of their time on evaluating them instead of doing research projects.

Solution: ACCES is built to solve this problem, it is a system for universities. ACCES enables academicians to create online homework/exams, and students to implement and submit their assessments online. When a submission is done, it will be evaluated by the system automatically. For that matter, students can see their results as soon as they submit their codes, and academicians do no longer have to put in time on grading process.

Future Work

The system is open to develop and plug new features. We listed some of the possible upcoming features.

- A plagiarism checker for detecting unintended behaviors of students.
- An annual rewind page that shows numerical statistics like number of codes submitted, time spent on ACCES etc.
- More detailed statistics table for task and assessment submissions including improvement progress.
- A graphical tool for displaying the test cases in a useful way like dynamic graph drawings.
- Integrating non-coding assessments like multiple choice quizzes.
- Separating courses by semesters.
- Letting lecturer/assistant to add an unchangeable file to project structure.
- Letting lecturer/assistant to limit the number of submissions per student.

Additionally, the main functionality of the system can be easily integrated into two different systems: an online interview platform for technology companies, and an online programming contest platform like Codeforces and Hackerrank.

7. References

1. "About Us" 2019 Online. https://www.hackerrank.com/about-us
2. "Contests" 2019 Online. https://www.hackerrank.com/contests
3. "Frequently Asked Questions" 2010 Online. http://codeforces.com/help
4. "CS Academy" Online. https://csacademy.com/about/
5. "calico 1.1.2" Online. https://pypi.org/project/calico/
6. "Gradescope" 2019 Online. https://www.gradescope.com
7. "draw.io" 2020 https://about.draw.io/
8. Osses, Felipe & Márquez, Gastón & Astudillo, Hernán. (2018). Exploration of academic and industrial evidence about architectural tactics and patterns in microservices. 256-257. 10.1145/3183440.3194958.
9. Gueye, Modou & Sarr, Idrissa & Ndiaye, Samba. (2009). Database replication in large scale systems: Optimizing the number of replicas. 3-9. 10.1145/1698790.1698794.
10. Taherizadeh, Salman & Grobelnik, Marko. (2019). Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. Advances in Engineering Software. 140. 10.1016/j.advengsoft.2019.102734.
11. Menychtas, Andreas & Konstanteli, Kleopatra. (2012). Fault detection and recovery mechanisms and techniques for service oriented infrastructures.
12. "Trello" 2020 https://trello.com/en/about
13. Ikonen, Marko & Pirinen, Elena & Fagerholm, Fabian & Kettunen, Petri & Abrahamsson, Pekka. (2011). On the Impact of Kanban on Software Project Work An Empirical Case Study Investigation. Proceedings - 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011. 305-314. 10.1109/ICECCS.2011.37.
14. "gitlab" 2014 https://about.gitlab.com/company/
15. Meyer, Mathias. (2014). Continuous Integration and Its Tools. Software, IEEE. 31. 14-16. 10.1109/MS.2014.58.