

PENGEMBANGAN SISTEM AUTO GRADING MENGUNAKAN PC PENGGUNA SEBAGAI WORKER

Laporan Tugas Akhir

Disusun sebagai syarat kelulusan tingkat sarjana

Oleh JAUHAR ARIFIN

NIM: 13515049



**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

15 Mei 2019

**PENGEMBANGAN SISTEM AUTO GRADING
MENGUNAKAN PC PENGGUNA SEBAGAI
WORKER**

Laporan Tugas Akhir

Oleh JAUHAR ARIFIN

NIM: 13515049

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung**

Bandung, 26 Desember 2018

Mengetahui,

Pembimbing,

Riza Satria Perdana, ST., MT.

NIP. 197006091995121002

ABSTRAK

PENGEMBANGAN SISTEM AUTO GRADING MENGUNAKAN PC PENGGUNA SEBAGAI WORKER

Oleh JAUHAR ARIFIN

NIM: 13515049

Competitive programming merupakan kompetisi di bidang *computer science* dimana peserta berlomba menyelesaikan persoalan dengan membuat program sesuai batasan yang ditentukan. Dalam menyelenggarakan kompetisi *competitive programming*, juri menggunakan *autograder* untuk melakukan penilaian. *Autograder* dijalankan pada komputer tertentu yang disebut *worker*. Untuk menjaga kinerja penilaian, diperlukan *worker* dalam jumlah yang besar sehingga diperlukan biaya yang besar pula.

Komputer peserta umumnya memiliki kemampuan yang cukup untuk melakukan penilaian jawaban. Pada tugas akhir ini, komputer peserta digunakan sebagai *worker*. Dengan menggunakan komputer peserta sebagai *worker*, kinerja penilaian dapat meningkat. Setiap peserta memiliki komputer dengan spesifikasi yang berbeda. Untuk menjaga keadilan, solusi peserta dan solusi juri akan dieksekusi pada *worker* dan dibandingkan hasilnya. Kerahasiaan penilaian dijaga dengan melakukan enkripsi pada tingkat aplikasi.

Pengujian tugas akhir ini dilakukan dengan menyimulasikan penilaian jawaban peserta oleh *autograder*. Pengujian dilakukan dengan membandingkan kinerja penilaian sistem yang dibangun dengan sistem *online judge* yang bersifat *open source* yaitu *DOMJudge*. Hasil pengujian menyatakan penilaian jawaban dengan memanfaatkan komputer peserta sebagai *worker* meningkatkan kinerja penilaian.

Kata kunci: *competitive programming, autograder, online judge*.

KATA PENGANTAR

Puji syukur kehadiran Allah SWT, atas limpahan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir dengan judul "Pengembangan Sistem Auto Grading Menggunakan PC Pengguna Sebagai Worker". Penulis mengucapkan terima kasih kepada semua pihak yang telah membantu pengerjaan tugas akhir ini. Ucapan terima kasih secara khusus penulis berikan kepada:

1. Dosen pembimbing Bapak Riza Satria Perdana, yang telah membimbing dan mengarahkan penulis dalam mengerjakan tugas akhir serta memberikan inspirasi dan pembelajaran yang sangat berarti.
2. Orang tua dan keluarga yang telah memberikan doa dan dukungannya dalam pengerjaan tugas akhir ini.
3. Arfinda Ilmania dan seluruh teman-teman Informatika 2015 yang telah memberikan masukan dan dukungan sehingga tugas akhir ini dapat diselesaikan.
4. Bapak dan ibu dosen informatika yang telah memberikan pembelajaran dan pengalaman yang tak ternilai selama empat tahun masa perkuliahan.

Penulis menyadari bahwa tugas akhir ini tidaklah sempurna. Untuk itu penulis ingin meminta maaf dan menerima kritik dan saran yang membangun dan dapat disampaikan melalui alamat email penulis yaitu: jauhararifin10@gmail.com.

Jauhar Arifin

Daftar Isi

Abstrak	i
Kata Pengantar	ii
Daftar Isi	iv
Daftar Gambar	v
Daftar Tabel	vi
I Pendahuluan	1
I.1 Latar Belakang	1
I.2 Rumusan Masalah	2
I.3 Tujuan	2
I.4 Batasan Masalah	3
I.5 Metodologi	3
I.6 Sistematika Pembahasan	3
II Studi Literatur	5
II.1 <i>Competitive Programming</i>	5
II.2 <i>Sistem Online Judge</i>	7
II.3 <i>Autograder</i>	8
II.4 <i>Virtual Machine</i>	9
II.5 <i>Containerization</i>	10
II.5.1 <i>Chroot</i>	11
II.5.2 <i>Cgroup</i>	12
II.5.3 <i>Namespace</i>	13
II.6 <i>WebAssembly</i>	13
III Analisis Masalah Dan Rancangan Solusi	15
III.1 <i>Sistem Autograder</i>	15
III.1.1 <i>Pengukuran Waktu Dan Memori</i>	16

III.1.2	<i>Load Balancing</i>	21
III.1.3	Evaluasi Jawaban Menggunakan Sandbox	23
III.1.4	Kompilasi Program	25
III.1.5	Pengiriman <i>Test-Case</i> Ke <i>Worker</i>	26
III.2	Sistem Manajemen Kompetisi	28
III.2.1	Komponen Pengguna	30
III.2.2	Komponen Soal	31
III.2.3	Komponen Jawaban	32
IV	Pengembangan Dan Pengujian	33
IV.1	<i>UGServer</i> (Sistem Manajemen Kompetisi)	33
IV.1.1	Manajemen Pengguna	35
IV.1.2	Manajemen Soal	37
IV.1.3	Manajemen Jawaban	39
IV.1.4	Penilaian Jawaban	41
IV.2	<i>UGSbox</i>	45
IV.3	<i>UGJob</i>	48
IV.3.1	Pembangkitan <i>Testcase</i>	50
IV.3.2	Penghitungan Waktu Dan <i>Memory</i> Solusi Juri	50
IV.3.3	Eksekusi Solusi Peserta	51
IV.3.4	Penilaian Keluaran Solusi Peserta	52
IV.4	<i>UGDesktop</i>	54
IV.5	Pengujian	55
V	Simpulan dan Saran	57
V.1	Simpulan	57
V.2	Saran	57

Daftar Gambar

II.2.1	Contoh Sistem <i>Online Judge</i> : Codeforces.	7
II.3.1	Proses penilaian program peserta (Danutama & Liem, 2013)	8
III.0.1	Arsitektur Sistem <i>Online Judge</i> Pada Saat Ini	15
III.1.1	Arsitektur Sistem <i>Online Judge</i> Yang Dibuat	16
III.1.2	Diagram Aktivitas Penilaian Solusi Peserta	20
III.1.3	<i>Push-Based Load Balancer</i>	22
III.1.4	<i>Pull-Based Load Balancer</i>	23
III.1.5	Diagram Aliran Data Pada Proses Penilaian Solusi Peserta	27
III.1.6	Alur Proses Penilaian Jawaban Peserta	28
III.2.1	Diagram <i>Use Case</i> Dari Sistem Manajemen Kompetisi	29
III.2.2	Diagram Komponen Sistem Manajemen Kompetisi	30
IV.1.1	Skema Basis Data Sistem Manajemen Kompetisi	35
IV.1.2	Skema Basis Data Sistem Manajemen Pengguna	36
IV.1.3	Skema Basis Data Sistem Manajemen Soal	38
IV.1.4	Skema Basis Data Sistem Manajemen Jawaban	40
IV.1.5	Skema Penilaian Jawaban Pengguna	41
IV.1.6	Contoh Proses Pembuatan <i>Grading</i>	44
IV.1.7	Contoh Proses Penilaian Oleh <i>Worker</i>	45
IV.4.1	Halaman Kompetisi Dari <i>UGDesktop</i>	54

Daftar Tabel

BAB I

PENDAHULUAN

I.1 Latar Belakang

Competitive programming merupakan salah satu cabang lomba yang cukup populer di bidang *computer science*. Pada kompetisi *competitive programming*, peserta diminta menyelesaikan persoalan terkait *computer science* yang diberikan oleh juri secara benar dan dalam waktu yang singkat. Beberapa instansi dan organisasi seringkali mengadakan kompetisi ini secara rutin. Perusahaan teknologi besar seperti Google dan Facebook pun seringkali mengadakan kompetisi *competitive programming* secara tahunan. Kompetisi *competitive programming* ditunjang dengan menggunakan sistem *online judge*. Sistem *online judge* tersebut biasanya berupa halaman *web* dimana peserta dapat melihat soal, membuat klarifikasi, mengirimkan jawaban dan melihat *scoreboard*. Sistem *online judge* yang populer pada saat ini adalah Codeforces, URI *Online Judge* (Bez, Tonin, & Rodegheri, 2014), Uva, dan SPOJ.

Di dalam sistem *online judge*, terdapat sistem *autograder* yang digunakan untuk menilai jawaban peserta. Jawaban peserta yang berupa *source code* dalam bahasa pemrograman tertentu akan dinilai kebenarannya oleh sistem *autograder* dengan cara melakukan kompilasi pada kode tersebut kemudian mengeksekusi program hasil kompilasi dengan *test-case* yang sudah disiapkan oleh juri atau pembuat soal. Menurut Fernando & Liem, 2014, cara tersebut disebut sebagai *black-box grading*. Dengan menggunakan *autograder*, penilaian jawaban peserta dapat dilakukan secara otomatis dan keterlibatan manusia menjadi lebih sedikit. Untuk meningkatkan jumlah jawaban peserta yang dapat dinilai dalam satuan waktu, biasanya juri menyiapkan lebih dari satu komputer yang menjalankan sistem *autograder*. Untuk menjalankan *autograder* pada lebih dari satu komputer, diperlukan komputer dengan spesifikasi yang sama untuk menjaga keadilan penilaian.

Saat ini, hampir semua kompetisi *competitive programming* menggunakan sistem au-

tograder. Kebanyakan dari kompetisi tersebut juga telah menggunakan lebih dari satu *autograder* untuk meningkatkan kinerja penilaian. Meskipun begitu, karena banyaknya peserta yang mengikuti kompetisi tersebut, seringkali jumlah *autograder* yang disediakan oleh juri kurang dan mengakibatkan jawaban peserta tidak dapat dinilai secara cepat. Penggunaan sistem *autograder* yang banyak juga menghabiskan banyak biaya karena perlu menyewa komputer dengan kinerja yang cukup tinggi untuk menjalankan sistem *autograder* tersebut. Oleh karena itu, diperlukan sistem penilaian baru yang dapat mengurangi biaya pengadaan infrastruktur guna menjalankan sistem *autograder*.

Dalam mengikuti kompetisi *competitive programming*, peserta umumnya menggunakan komputer pribadinya untuk menulis program yang digunakan untuk menyelesaikan soal yang diberikan. Setiap komputer yang digunakan oleh peserta umumnya memiliki spesifikasi yang cukup untuk melakukan kompilasi pada *source code* yang ditulis oleh peserta dan melakukan eksekusi program hasil kompilasi tersebut. Oleh sebab itu, komputer peserta berpotensi untuk menjadi infrastruktur yang dapat digunakan untuk melakukan penilaian program oleh *autograder*.

I.2 Rumusan Masalah

Masalah yang diselesaikan dalam tugas akhir ini adalah:

1. Sistem *autograder* yang sering digunakan saat ini melakukan penilaian terhadap kode peserta dengan melakukan kompilasi dan eksekusi pada komputer yang disediakan oleh juri. Komputer yang digunakan oleh peserta kompetisi *competitive programming* memiliki kemampuan untuk menjalankan sistem *autograder*. Bagaimana memanfaatkan komputer peserta tersebut untuk menjalankan sistem *autograder* untuk menilai jawaban peserta saat kompetisi sedang berlangsung.
2. Dalam menjalankan sistem *autograder*, aspek keamanan, keadilan dan kinerja perlu diperhatikan. Bagaimana menjaga keamanan, keadilan dan kinerja dari sistem *autograder* yang berjalan pada komputer peserta.

I.3 Tujuan

Tujuan dari tugas akhir ini adalah meningkatkan kinerja penilaian jawaban peserta pada kompetisi *competitive programming* dengan menciptakan sistem *autograder* yang dapat berjalan pada komputer peserta.

I.4 Batasan Masalah

Pada pengerjaan tugas akhir ini, diasumsikan seluruh peserta yang menggunakan perangkat lunak ini memiliki komputer dengan spesifikasi yang cukup untuk menjalankan sistem *autograder* dan memiliki sistem operasi berbasis Linux. Perangkat lunak yang dibangun untuk menjalankan sistem *autograder* hanya mencakup sistem *autograder* untuk domain *competitive programming* saja. Selain itu, peserta diasumsikan tidak melakukan serangan yang berbentuk *reverse engineering*.

I.5 Metodologi

Metodologi yang dilakukan dalam pengerjaan Tugas Akhir ini adalah:

1. Menganalisis dan mendesain sistem *autograder*

Pada tahap ini dilakukan analisis terhadap teknik pembuatan sistem *autograder* beserta aspek-aspek yang perlu diperhatikan dalam melakukan pengembangan sistem *autograder*. Selain itu, pada tahap ini juga dihasilkan metode yang akan digunakan untuk mengembangkan sistem *autograder* yang dapat berjalan pada komputer peserta.

2. Desain dan analisis perangkat lunak

Pada tahap ini dilakukan analisis terhadap kebutuhan perangkat lunak beserta membuat desain perangkat lunak yang akan diimplementasikan.

3. Implementasi pengembangan perangkat lunak

Pada tahap ini, implementasi dari pengembangan perangkat lunak dilakukan berdasarkan desain yang sudah dibuat pada tahap sebelumnya.

4. Pengujian

Perangkat lunak yang telah dihasilkan diuji pada tahap ini sesuai dengan kebutuhan yang telah didefinisikan.

5. Penarikan kesimpulan

Pada tahap ini, hasil pengujian digunakan untuk melakukan penarikan kesimpulan.

I.6 Sistematika Pembahasan

Sistematika pembahasan dari Laporan Tugas Akhir ini adalah sebagai berikut.

1. BAB I PENDAHULUAN berisi penjelasan mengenai latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, serta sistematika pembahasan tugas akhir.
2. BAB II STUDI LITERATUR berisi hasil studi literatur berupa teori yang mendasai rancangan penyelesaian tugas akhir. Landasan teori terdiri dari kompetisi *competitive programming*, sistem *online judge*, *autograder*, *virtual machine*, *containerization* dan *webassembly*.
3. BAB III ANALISIS MASALAH DAN RANCANGAN SOLUSI berisi persoalan yang muncul ketika menggunakan komputer peserta sebagai *worker* dari *autograder* beserta penyelesaian dari permasalahan tersebut.
4. BAB IV PENGEMBANGAN DAN PENGUJIAN berisi bahasa pemrograman, *tools* dan komponen-komponen yang dibuat dalam mengimplementasikan rancangan solusi pada BAB III.
5. BAB V SIMPULAN DAN SARAN berisi kesimpulan dan saran dari proses dan hasil pengembangan dan pengujian perangkat lunak.

BAB II

STUDI LITERATUR

Pada bab ini dipaparkan hasil studi literatur terkait dengan kompetisi *competitive programming*. Studi literatur yang dilakukan adalah terkait sistem yang digunakan dalam menyelenggarakan kompetisi *competitive programming*, *online judging system*, *auto-grader* dan metode yang digunakan dalam melakukan grading.

II.1 *Competitive Programming*

Competitive programming merupakan sebuah kompetisi dimana peserta dari kompetisi tersebut diminta menyelesaikan suatu permasalahan pada bidang *computer science* secara cepat dan tepat. Pada kompetisi *competitive programming*, peserta akan diberikan permasalahan *computer science* yang sudah pernah diselesaikan dan bukan permasalahan riset yang solusinya masih belum ditemukan (S. Halim & F. Halim, 2013). Untuk setiap persoalan yang diberikan, peserta diminta untuk menyelesaikan masalah tersebut dengan membuat program dalam bahasa pemrograman yang diizinkan oleh juri. Program yang dibuat oleh peserta harus memenuhi batasan yang dibuat oleh juri seperti waktu eksekusi, kebutuhan memori, dan ukuran program.

Pada kompetisi *competitive programming*, program yang telah dibuat oleh peserta akan dinilai dengan suatu metode tertentu. Umumnya penilaian yang dilakukan mencakup kebenaran program dan waktu pengumpulan program, akan tetapi terdapat beberapa metode penilaian lain yang dapat dilakukan. Beberapa standar metode telah digunakan untuk melakukan penilaian jawaban dalam kompetisi *competitive programming*, diantaranya adalah standar IOI dan ICPC. Beberapa kompetisi *competitive programming* diselenggarakan oleh suatu organisasi, contohnya adalah ICPC yang diselenggarakan oleh ICPC Foundation (ICPC Foundation, 2018a), dan IOI (International Olympiad in Informatics) yang diselenggarakan oleh IOI Community (IOI Organization, 2017b). Selain itu, terdapat juga kompetisi *competitive programming* yang diselenggarakan

oleh suatu perusahaan seperti Google Code Jam yang diselenggarakan oleh Google dan Facebook HackerCup yang diselenggarakan oleh Facebook.

Kompetisi *competitive programming* pada tingkat perguruan tinggi biasanya mengikuti standar ICPC. Beberapa kompetisi yang mengikuti standar ini antara lain adalah Gemastik, Compfest, Vocomfest, INC, dan ACM-ICPC. Pada standar ICPC, setiap peserta akan bekerja dalam sebuah tim yang terdiri dari tiga peserta. Tiap tim akan diberikan soal yang sama. Seluruh tim akan mulai mengerjakan soal secara bersama-sama. Program yang telah dibuat oleh sebuah tim akan dikirimkan ke sistem penilaian untuk dinilai. Penilaian dilakukan secara otomatis oleh sistem yang disebut *online judge*. Pada sistem *online judge*, penilaian dilakukan oleh subsistem yang bernama *autograder* yang berjalan pada sistem tersebut. *Autograder* akan menjalankan program yang dikirimkan oleh peserta dan menentukan apakah program tersebut benar atau salah. Setiap program yang dikirimkan oleh peserta hanya dapat bernilai benar atau salah. Setiap tim yang mengirimkan jawaban salah akan mendapatkan penalti waktu. Nilai total dari sebuah tim dihitung dari jumlah soal yang berhasil dikerjakan oleh tim tersebut. Jika terdapat dua tim yang menyelesaikan soal dengan jumlah yang sama, maka jumlah penalti akan dihitung untuk menentukan tim yang nilainya lebih tinggi (ICPC Foundation, 2018b).

Selain standar ICPC, terdapat standar lain yang biasa digunakan untuk tingkat sekolah menengah atas yaitu standar IOI. Pada standar IOI, setiap peserta bekerja secara individu dan mendapatkan soal yang sama. Berbeda dengan standar ICPC, pada standar IOI setiap soal memiliki beberapa *subtask* dengan nilai tertentu. Peserta dapat menyelesaikan soal secara parsial dan mendapatkan nilai berdasarkan total dari *subtask* yang berhasil diselesaikan dengan benar pada soal tersebut (IOI Organization, 2017a).

Terdapat jenis *competitive programming* lain yang tidak memiliki standar tertentu, misalnya Google Code Jam dan Codeforces. Pada kompetisi Google Code Jam, sistem tidak melakukan penilaian dengan menjalankan program yang dibuat oleh peserta, melainkan hanya meminta jawaban dari peserta dalam bentuk *file* keluaran yang sudah dihasilkan oleh program peserta yang dijalankan di komputer peserta sendiri. Pada kompetisi Codeforces, sistem penilaian memiliki banyak perbedaan dengan sistem penilaian lain. Pada kompetisi ini, setiap soal memiliki nilai yang berbeda sesuai dengan tingkat kesulitannya dan nilainya akan terus berkurang selama kompetisi berlangsung. Selain itu, pada kompetisi Codeforces peserta dapat melakukan *hack* pada program yang telah dikirimkan peserta lain (Mirzayanov, 2011).

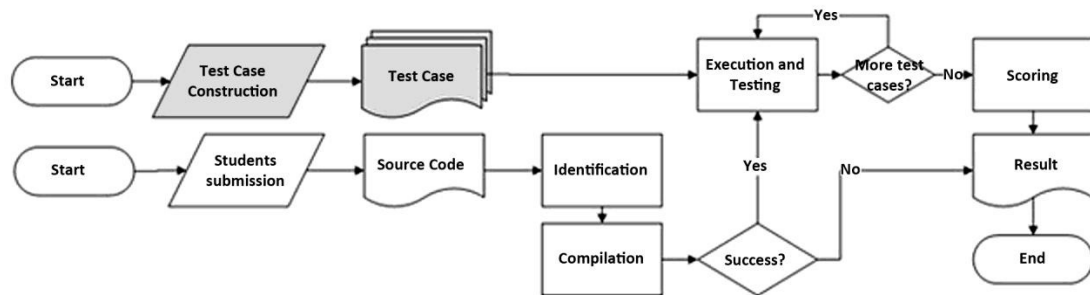
II.2 Sistem *Online Judge*

Online judge merupakan suatu *platform* yang digunakan untuk menyelenggarakan kompetisi *competitive programming*. Peserta kompetisi menggunakan sistem *online judge* untuk mengakses soal dan mengirimkan jawaban atau program yang telah dibuat. Sistem *online judge* akan melakukan kompilasi pada kode yang dikirimkan peserta lalu mengevaluasi program yang dihasilkan dengan *test-case* tertentu untuk menilai jawaban peserta tersebut (Wasik et al., 2018). Selain itu, sistem *online judge* juga memiliki beberapa fungsi lain diantaranya adalah menampilkan *scoreboard* dan memudahkan peserta melakukan klarifikasi pada soal. Umumnya sistem *online judge* memiliki antar muka berupa halaman *web* yang dapat digunakan oleh peserta untuk membaca soal dan mengirimkan jawaban dari soal tersebut. Gambar II.2.1 memperlihatkan contoh halaman soal dari sistem *online judge* yang cukup populer yaitu Codeforces.

The screenshot displays the Codeforces website interface. At the top, the Codeforces logo is visible, along with user information for 'jauhararifin' and a 'Logout' link. A navigation bar includes links for HOME, TOP, CONTESTS, GYM, PROBLEMS (selected), GROUPS, RATING, API, HELP, FORETHOUGHT FUTURE CUP, and CALENDAR. Below the navigation bar, the 'PROBLEMS' section is active, showing a list of problems. The selected problem is 'A. Theatre Square', which is part of the 'Codeforces Beta Round #1'. The problem description states: 'Theatre Square in the capital city of Berland has a rectangular shape with the size $n \times m$ meters. On the occasion of the city's anniversary, a decision was taken to pave the Square with square granite flagstones. Each flagstone is of the size $a \times a$. What is the least number of flagstones needed to pave the Square? It's allowed to cover the surface larger than the Theatre Square, but the Square has to be covered. It's not allowed to break the flagstones. The sides of flagstones should be parallel to the sides of the Square.' The input and output specifications are provided, along with an example input and output. The example input is '6 6 4' and the example output is '4'. On the right side of the problem page, there are buttons for 'Finished', 'Practice', and 'Start virtual contest'. There is also a section for 'Clone Contest to Mashup' and a 'Submit?' section with a language dropdown (set to 'GNU G++11 5.1.0') and a 'Submit' button.

Gambar II.2.1: Contoh Sistem *Online Judge* : Codeforces.

Beberapa organisasi memiliki *online judge* yang secara publik dapat diakses, dianta-



Gambar II.3.1: Proses penilaian program peserta (Danutama & Liem, 2013)

ranya adalah: URI *Online Judge*, TLX, Codeforces, Uva, SPOJ, dan lain sebagainya. *Online judge* yang bersifat publik ini biasanya memiliki beberapa soal-soal yang dapat digunakan untuk latihan dan dapat dikerjakan tanpa harus berkompetisi dengan peserta lain. Pada URI *Online Judge* terdapat fitur tambahan seperti forum dan *rewarding system* (Bez, Tonin, & Rodegheri, 2014). Terdapat beberapa sistem *online judge* yang bersifat *open source* seperti Mooshak, Judgels, dan DomJudge. Sistem *online judge* yang bersifat *open source* biasanya digunakan oleh beberapa instansi seperti perguruan tinggi untuk membuat kompetisi *competitive programming* yang bersifat tertutup seperti Gemastik, Compfest, Vocompfest, Arkavidia, dan INC.

II.3 Autograder

Pada *competitive programming*, *autograder* merupakan suatu sistem yang digunakan oleh *online judge* untuk melakukan kompilasi, eksekusi dan menilai *source code* (Danutama & Liem, 2013). *Autograder* digunakan dalam sistem *online judge* untuk menilai kebenaran suatu *source code*. Umumnya *autograder* akan melakukan kompilasi pada *source code* yang dikirimkan oleh peserta pada kompetisi *competitive programming*. Hasil kompilasi dari kode peserta tersebut akan diuji dengan menggunakan *test-case* rahasia yang telah dibuat oleh juri atau *problem setter*. Setelah melalui pengujian, program dinilai kebenarannya berdasarkan hasil pengujian tersebut. Menurut Fernando & Liem, 2014, cara tersebut disebut sebagai *black-box grading*. Gambar II.3.1 menjelaskan alur penilaian sebuah *source code* yang dikirimkan peserta.

Seluruh proses penilaian sebuah *source code* peserta membutuhkan waktu tiga menit jika dilakukan secara manual, sedangkan hanya membutuhkan waktu sepuluh detik dengan menggunakan *autograder* (Danutama & Liem, 2013). Dengan menggunakan *autograder*, peserta kompetisi *competitive programming* dapat menerima *feedback* dengan lebih cepat dan mengurangi pekerjaan yang harus dilakukan oleh juri.

Dalam menyelenggarakan kompetisi *competitive programming*, juri umumnya menggunakan banyak *autograder* untuk melakukan penilaian jawaban peserta. Dengan menggunakan banyak *autograder*, proses penilaian dapat dilakukan dengan lebih cepat. Untuk menjaga keadilan penilaian, satu buah *autograder* umumnya hanya dapat menilai satu jawaban peserta dalam satu waktu. Jika peserta kompetisi *competitive programming* sangat banyak, maka diperlukan *autograder* yang banyak pula untuk dapat melakukan penilaian jawaban peserta dengan cepat.

Terdapat banyak hal yang harus diperhatikan dalam membangun sistem *online judge* dengan menggunakan *autograder*. Salah satu permasalahan yang harus dihadapi dalam membangun *autograder* adalah masalah keamanan sistem. Sistem *autograder* harus dapat bertahan terhadap serangan yang mungkin dilakukan oleh peserta. Beberapa serangan yang mungkin dilakukan oleh peserta adalah mengirimkan *source code* yang memiliki waktu kompilasi yang sangat lama dan membebani sistem, membuat kode yang dapat mengubah atau merusak lingkungan *autograder*, dan mengakses *resource* dari *autograder* yang tidak diizinkan oleh sistem (Wasik et al., 2018). Metode *sandboxing* dapat dilakukan untuk mengatasi masalah keamanan tersebut. *Sandboxing* merupakan teknik untuk mengisolasi suatu eksekusi program sehingga tidak mengganggu lingkungannya. Metode *sandboxing* yang populer pada saat ini antara lain adalah dengan memanfaatkan teknologi *virtualization* seperti KVM, atau teknologi *container* seperti LXC dan Docker.

Hal lain yang perlu diperhatikan dalam membangun *autograder* adalah aspek keadilan. Waktu eksekusi program perlu diukur dengan tepat untuk menciptakan sistem yang adil. Waktu eksekusi program yang sama dengan input yang sama dapat memiliki nilai yang berbeda bergantung beberapa faktor seperti kecepatan CPU, ukuran RAM, dan lain sebagainya. Waktu pengukuran sebuah eksekusi program dalam sistem *autograder* biasanya dilakukan dalam hitungan milidetik. Beberapa metode yang dapat digunakan untuk mengukur waktu pemrosesan adalah dengan melakukan analisis pada *hardware performance*, *code instrumentation*, atau *code sampling* (Wasik et al., 2018). Umumnya juri menjaga keadilan penilaian dengan menggunakan spesifikasi komputer yang sama untuk menjalankan banyak sistem *autograder*. Komputer dengan spesifikasi yang sama diharapkan dapat menilai jawaban peserta secara adil.

II.4 *Virtual Machine*

Virtual machine merupakan teknik *sandboxing* yang memberikan virtualisasi pada tingkat *hardware*. Dengan menggunakan *virtual machine*, pengguna dapat menggu-

nakan komputernya untuk membuat suatu lingkungan yang terisolasi dan seakan-akan merupakan komputer baru yang terpisah dari komputer pengguna. Hal tersebut dapat dilakukan karena adanya *hypervisor* yang digunakan untuk melakukan virtualisasi *hardware* dari suatu komputer. Dengan menggunakan *virtual machine*, pengguna dapat menjalankan lebih dari satu sistem operasi berbeda di dalam komputer yang sama. Pada sistem operasi yang berbasis Linux, terdapat suatu fitur yang bernama KVM yang memungkinkan Linux menjadi *hypervisor* dan menjalankan *guest OS* di dalam proses Linux (Felter et al., 2015).

Proses (program yang berjalan) di dalam *virtual machine* terisolasi dari proses yang ada di luar-nya. *Resource* seperti memori dan CPU pada proses di *virtual machine* juga dapat dibatasi. Proses di atas *virtual machine* akan dikelola oleh sistem operasi yang berjalan di *virtual machine* tersebut. Dengan menggunakan KVM, pengguna dapat menjalankan program yang arsitekturnya benar-benar berbeda dengan Linux seperti program berarsitektur Windows. Untuk menjalankan suatu program pada lingkungan *virtual machine*, diperlukan adanya sistem operasi yang mengelola keberjalanan program tersebut. Menjalankan sebuah program pada *virtual machine* dapat menjadi sangat berat karena perlu adanya sistem operasi sendiri yang mengakibatkan adanya pekerjaan tambahan untuk *booting*, manajemen memori, manajemen proses, *scheduling*, dan lain sebagainya.

Virtual machine tidak cocok untuk digunakan oleh *autograder* karena terlalu banyak pekerjaan tambahan yang diperlukan dan memberatkan sistem. Untuk mengevaluasi *source code* peserta, tidak perlu menggunakan sistem operasi sendiri yang berbeda dengan sistem operasi yang menjalankan sistem *autograder*. *Virtual machine* lebih cocok digunakan untuk melakukan pengujian program yang membutuhkan berbagai macam jenis sistem operasi. Selain itu, *virtual machine* juga kerap digunakan dalam pembuatan sistem IaaS (*Infrastructure as a service*) seperti Amazon EC2, Digital Ocean Droplet dan Google Cloud Compute Engine.

II.5 Containerization

Secara singkat, *container* merupakan teknologi virtualisasi proses pada tingkat sistem operasi. Teknik *containerization* berbeda dengan teknik virtualisasi menggunakan *hypervisor* yang bekerja pada tingkat *hardware* (Merkel, 2014). *Container* bekerja seperti *virtual machine* yang dapat memberikan isolasi terhadap program yang berjalan. Pada *virtual machine*, virtualisasi terjadi pada tingkat *hardware* sehingga pengguna perlu memasang sistem operasi pada lingkungan yang terisolasi untuk dapat menjalan-

kan program. Pada *container*, pengguna tidak perlu memasang sistem operasi karena virtualisasi terjadi pada tingkat *software* sehingga sistem operasi yang berjalan pada komputer pengguna dapat digunakan di dalam *container*. Hal tersebut membuat *container* lebih ringan dibandingkan *virtual machine* karena tidak ada kerja tambahan untuk menjalankan sistem operasi baru.

Pada Linux, teknologi *container* mungkin dilakukan karena adanya beberapa fitur dari Linux yaitu: *chroot*, *cgroup* dan *namespace*. *Chroot* dapat memberikan isolasi *file system* terhadap suatu proses pada Linux. *Cgroup* dapat memberikan batasan *resource* kepada suatu proses di Linux. *Namespace* dapat memberikan isolasi pada proses sehingga proses dalam suatu lingkungan tidak dapat mengetahui adanya proses lain di lingkungan yang berbeda. Terdapat beberapa perangkat lunak yang menawarkan teknologi *containerization* ini, yang populer diantaranya adalah Docker dan LXC. Dengan menggunakan Docker atau LXC, pengguna dapat membuat suatu lingkungan yang terisolasi untuk menjalankan suatu program tertentu tanpa mengganggu lingkungan di luar-nya. Docker memberikan API yang lebih *high-level* dibandingkan dengan LXC, dan memiliki banyak fitur yang dapat digunakan untuk mengelola *container*. Docker sudah banyak digunakan sebagai *platform* untuk menjalankan dan mendistribusikan aplikasi.

Teknologi *container* ini dapat digunakan untuk melakukan penilaian terhadap kode program yang dikirimkan oleh peserta kompetisi *competitive programming*. Dengan menggunakan *container*, program yang dijalankan dapat diisolasi sehingga tidak membahayakan lingkungan di luar-nya. Selain itu, *resource* seperti CPU, memori, *storage*, dan IO pada *container* dapat dibatasi sehingga tidak mengganggu program lain yang sedang berjalan pada lingkungan di luar *container* (Merkel, 2014).

II.5.1 *Chroot*

Chroot merupakan *system call* pada sistem operasi yang berbasis Linux atau Unix. *System call* ini dapat mengubah *root file system* dari suatu proses ke direktori target tertentu. Program yang dijalankan seakan-akan memiliki direktori *root* sebagai direktori yang ditentukan pada waktu pemanggilan *system call chroot*. Dengan menggunakan cara ini, program yang dijalankan dalam *chroot* hanya dapat mengakses *file system* yang berada pada direktori target saja dan mengurangi kemungkinan serangan yang mungkin dilakukan pada *file system* asli (P., 2003). Dengan menggunakan *chroot*, pengguna dapat mengatur *library* apa saja yang dapat digunakan oleh program yang berjalan dalam lingkungan *chroot*. Hal ini dapat menjadi merepotkan karena pengguna perlu memasukkan semua *library* yang dibutuhkan ke dalam *file system chroot*. Tekno-

logi *container* memanfaatkan *chroot* untuk mengisolasi *file system* yang dapat diakses oleh program yang berjalan pada *container*.

Meskipun *chroot* memberikan penghalang pada aplikasi yang berada dalam lingkungan *chroot* untuk mengakses *file system* yang berada di luar lingkungan direktori target, *chroot* masih memiliki kelemahan. Jika aplikasi yang berjalan pada lingkungan *chroot* memiliki *root permission*, maka aplikasi tersebut dapat dengan mudah keluar dari lingkungan *chroot*. Terdapat beberapa cara untuk meningkatkan keamanan *chroot* sehingga program yang berjalan di dalam *chroot* sulit untuk keluar dari lingkungan tersebut.

Salah satu cara untuk meningkatkan keamanan pada lingkungan *chroot* adalah dengan memasukkan file yang hanya dibutuhkan oleh program saja. Sebagai contoh, jika program membutuhkan daftar *user* yang berada dalam sistem, di dalam lingkungan *chroot* perlu ada file */etc/passwd* untuk mendapatkan informasi ini. Akan tetapi, file */etc/shadow* tidak perlu dimasukkan ke dalam *file system chroot* karena mengandung informasi rahasia yang tidak diperlukan oleh program yang berada dalam lingkungan *chroot*.

Cara kedua untuk meningkatkan keamanan lingkungan *chroot* adalah dengan tidak memberikan akses *root* kepada program yang berjalan di dalam *chroot*. Terdapat beberapa serangan yang memungkinkan program dalam lingkungan *chroot* untuk keluar dari lingkungan *chroot* jika memiliki akses *root*. Untuk menghindari jenis serangan ini, sebaiknya program yang berjalan di dalam lingkungan *chroot* dibuat agar tidak mungkin mendapatkan akses *root*. Tidak memberikan akses *root* kepada program akan mengurangi kemungkinan adanya serangan. Akan tetapi, meskipun tidak memberikan akses *root*, masih terdapat beberapa serangan yang memungkinkan program membangkitkan akses *root* tanpa memiliki akses *root* pada saat dijalankan.

Untuk meningkatkan keamanan pada lingkungan *chroot*, sebaiknya tidak memasukkan *hard link* ke dalam *file system chroot*. *Hard link* yang mengacu pada file di luar lingkungan *chroot* akan mengurangi keamanan *chroot* karena memungkinkan program yang berada di dalam lingkungan *chroot* untuk mengakses file yang berada di luar lingkungan *chroot*.

II.5.2 Cgroup

Cgroup merupakan fitur pada Linux yang dapat digunakan untuk membatasi *resource* yang digunakan oleh suatu kelompok program (Felter et al., 2015). *Resource* yang dimaksud di sini adalah *CPU usage*, *memory usage* dan *disk IO*. Dengan menggunakan *cgroup*, sebuah program yang dibatasi *resource*-nya tidak akan mengganggu program

lain dengan cara menghabiskan *resource*. Fitur *cgroup* dimanfaatkan oleh teknologi *container* untuk membatasi *resource* pada sebuah *container* sehingga tidak membebani program lain di luar *container*. Pada *container*, umumnya *cgroup* digunakan untuk membatasi CPU dan memori yang digunakan oleh program-program yang berjalan di dalam *container*. *Cgroup* dapat memberhentikan program di dalam *container* jika telah menggunakan *resource* yang berlebihan. Selain itu, *cgroup* juga dapat digunakan untuk mengukur penggunaan *resource* oleh suatu proses yang berjalan pada komputer. Hal tersebut dapat digunakan untuk mengukur waktu eksekusi program peserta.

II.5.3 *Namespace*

Dengan menggunakan *chroot* dan *cgroup*, sebuah program dapat diisolasi sehingga memiliki *resource* dan *file system* sendiri yang terpisah dari program-program lain yang berjalan pada sistem operasi *host*. Yang dimaksud sistem operasi *host* disini adalah sistem operasi yang digunakan oleh pengguna tanpa adanya isolasi *container*. Meskipun *resource* dan *file system* dari proses sudah berhasil diisolasi, proses ini masih dapat melihat proses apa saja yang sedang berjalan pada sistem operasi *host* karena program yang berjalan di dalam *chroot* dan *cgroup* masih menggunakan sistem operasi yang sama dan menggunakan *kernel* yang sama. Linux memiliki fitur *namespace* yang memberikan isolasi kepada proses terhadap proses lain pada lingkungan yang berbeda. Dengan menggunakan *namespace*, sebuah proses di suatu *namespace* tertentu dan hanya dapat mengetahui proses lain yang berada di dalam *namespace* yang sama. *Container* memanfaatkan *namespace* sehingga program yang berjalan di dalam *container* hanya mengetahui proses lain yang berada di dalam *container* tersebut dan tidak dapat mengetahui proses pada sistem operasi *host*-nya (Felter et al., 2015).

Selain isolasi program, *namespace* juga memberikan isolasi pada hal lain seperti *network*, *mount*, *cgroup*, *user*, dan lain sebagainya. Dengan memberikan isolasi pada *network*, program yang berada di dalam *container* dapat membuka *port* yang sama seperti program lain yang berada di *container* lain. Dengan menggunakan isolasi pada *user*, *user* yang ada di dalam *container* tidak dapat diketahui oleh *container* lain.

II.6 WebAssembly

WebAssembly merupakan API Web yang memungkinkan *browser* untuk menjalankan *low-level code* secara aman. Dulunya Javascript merupakan satu-satunya bahasa yang didukung secara *native* oleh *web*, akan tetapi semakin berkembangnya teknologi *web*

kebutuhan akan kinerja dari *web* semakin tinggi. Javascript yang merupakan *interpreted language* tidak dapat memberikan kinerja yang tinggi seperti bahasa-bahasa yang dikompilasi menjadi *low-level code*. WebAssembly memberikan solusi yang memungkinkan *browser* untuk menjalankan *low-level code* pada sebuah sistem yang terisolasi dan aman. WebAssembly didesain untuk digunakan bersama-sama dengan Javascript untuk mengembangkan aplikasi *web*. Sebuah aplikasi *web* dapat memanfaatkan WebAssembly untuk meningkatkan kinerja dan memanfaatkan Javascript untuk fleksibilitas (Stevenson, 2018).

Dengan menggunakan WebAssembly, *browser* dapat membuat sebuah lingkungan yang terisolasi untuk menjalankan *low-level code*. Lingkungan yang digunakan untuk menjalankan kode WebAssembly memiliki beberapa batasan seperti jumlah memori yang bisa digunakan, *file* yang bisa diakses dan lain sebagainya. Umumnya kode WebAssembly hanya dapat melakukan apa yang dapat dilakukan oleh *web*. *Web* tidak dapat mengakses sembarang *file* yang berada pada komputer, begitu juga dengan WebAssembly. *Web* tidak dapat melakukan beberapa *system call*, begitu juga WebAssembly. Meskipun WebAssembly dapat memberikan isolasi pada program yang berjalan, pembatasan *resource* pada WebAssembly masih sulit dilakukan dan tidak semudah pembatasan *resource* pada *container* ataupun *virtual machine*.

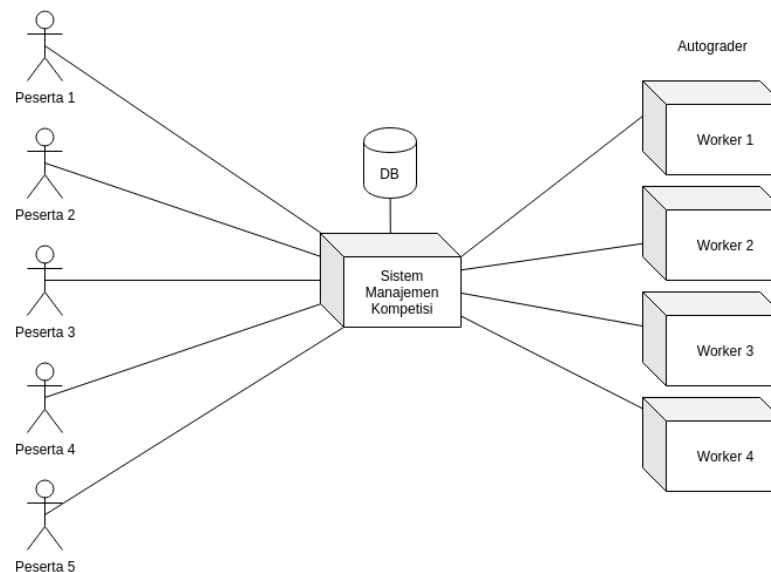
Kode WebAssembly berupa *binary code* yang dapat dibuat dengan cara melakukan kompilasi dari beberapa bahasa seperti C, C++ dan Rust ke dalam format *wasm*. WebAssembly masih baru dan belum banyak bahasa pemrograman yang dapat dikompilasi menjadi WebAssembly. Bahasa pemrograman yang menggunakan *interpreter* atau *runtime environment* seperti Python dan Java masih sulit dijalankan di dalam WebAssembly.

Dalam sistem *autograder*, WebAssembly tidak cocok digunakan karena berbagai alasan, yaitu: masih sedikitnya bahasa pemrograman yang dapat dikompilasi menjadi kode WebAssembly, sulitnya membatasi *resource* pada WebAssembly, sulitnya melakukan perhitungan waktu, dan sulitnya menjaga keamanan. *Browser* yang ada pada saat ini memiliki kemampuan untuk mengubah kode Javascript yang berjalan pada *browser* tersebut, hal ini mengakibatkan munculnya celah keamanan pada WebAssembly jika digunakan untuk mengevaluasi kode peserta.

BAB III

ANALISIS MASALAH DAN RANCANGAN SOLUSI

Dalam mengadakan kompetisi *competitive programming* diperlukan sebuah sistem *online judge*. Sistem *online judge* memerlukan dua buah komponen utama yaitu *autograder* dan sistem manajemen kompetisi. Sistem manajemen kompetisi memberikan layanan yang berhubungan dengan kompetisi seperti melihat soal, mengirim jawaban, membuat klarifikasi, dan melihat *scoreboard*. Sistem *autograder* berfungsi untuk melakukan penilaian terhadap jawaban yang telah dikirim secara *realtime*. Gambar III.0.1 menggambarkan arsitektur sistem *online judge* yang saat ini sering digunakan.

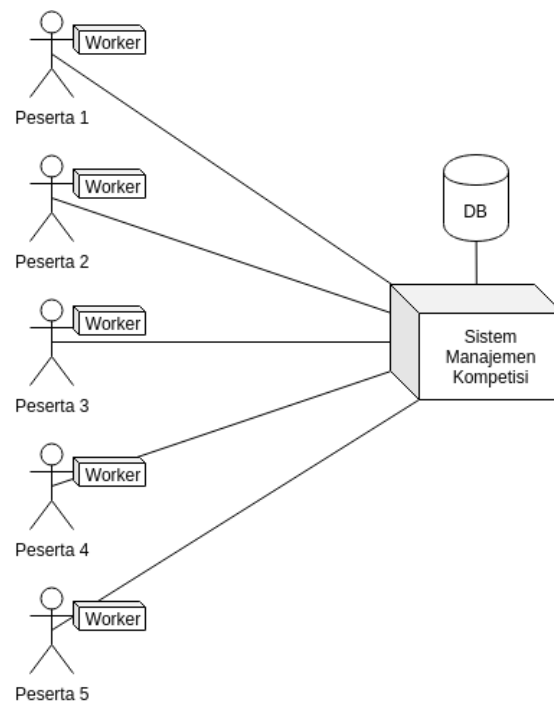


Gambar III.0.1: Arsitektur Sistem *Online Judge* Pada Saat Ini

III.1 Sistem *Autograder*

Sistem manajemen kompetisi memerlukan *autograder* untuk menilai jawaban peserta kompetisi secara otomatis. Sistem *online judge* yang sering digunakan saat ini menggunakan *autograder* yang dipasang oleh juri pada beberapa komputer yang sudah di-

sediakan oleh juri. Pada tugas akhir ini, diciptakan sistem *autograder* yang dapat berjalan pada komputer peserta. Komputer peserta akan bertindak sebagai *worker* yang menjalankan sistem *autograder*. *Worker* adalah komputer yang digunakan oleh sistem *autograder* untuk melakukan kompilasi dan eksekusi terhadap jawaban peserta. Dalam tugas akhir ini, komputer peserta adalah *worker* dari sistem *autograder*. Terdapat beberapa aspek yang perlu diperhatikan dalam mengembangkan sistem *autograder* seperti pengukuran waktu dan memori, *load balancing*, evaluasi jawaban peserta, dan pengiriman *test-case* ke *worker*. Gambar III.1.1 menggambarkan arsitektur sistem *on-line judge* yang dibuat pada tugas akhir ini.



Gambar III.1.1: Arsitektur Sistem *Online Judge* Yang Dibuat

III.1.1 Pengukuran Waktu Dan Memori

Komputer peserta memiliki spesifikasi yang berbeda-beda. Perbedaan spesifikasi tersebut menimbulkan beberapa perbedaan ketika sebuah program yang sama dieksekusi. Program yang sama dengan masukan yang sama dapat berjalan dengan waktu dan memori yang berbeda pada komputer yang berbeda. Komputer dengan *clock-speed* yang lebih tinggi akan menjalankan program dengan lebih cepat. Komputer dengan *core* yang lebih banyak juga akan menjalankan program parallel dengan lebih cepat. Selain itu, ukuran *cache* dari komputer juga memengaruhi kecepatan waktu eksekusi. Kebutuhan memori dari suatu program juga ditentukan oleh spesifikasi komputer. Komputer dengan arsitektur 64-bit umumnya memerlukan memori yang lebih besar dibanding-

kan arsitektur 32-bit. Oleh karena itu, program yang dieksekusi pada komputer yang berbeda akan memerlukan waktu dan memori yang berbeda pula.

Penilaian jawaban peserta memerlukan metode pengukuran waktu dan memori yang adil sehingga program peserta yang benar akan dianggap benar oleh setiap *worker*. Begitu juga program peserta yang salah akan dianggap salah oleh setiap *worker*. Pada bab ini, dipaparkan beberapa teknik yang dapat digunakan untuk mengukur waktu eksekusi jawaban peserta.

III.1.1.1 Spesifikasi CPU dan Sistem Operasi

Kecepatan eksekusi sebuah program bergantung pada *clock-speed* dan ukuran *cache* dari CPU. Jika suatu proses hanya diberikan satu buah *core* dari CPU, maka jumlah *core* dari CPU dapat dianggap tidak memengaruhi kecepatan eksekusi proses tersebut. Kecepatan eksekusi program dapat diukur berdasarkan spesifikasi CPU dari komputer yang menjalankannya. Program yang dijalankan pada komputer dengan *clock-speed* dan ukuran *cache* yang rendah akan diberikan batasan waktu yang lebih lama. Sedangkan program yang dijalankan pada komputer dengan *clock-speed* dan ukuran *cache* yang lebih tinggi akan diberikan batasan waktu yang lebih singkat. Kebutuhan memori dari program yang dijalankan pada suatu komputer dipengaruhi oleh arsitektur komputer tersebut. Komputer peserta dapat memiliki arsitektur yang berbeda-beda. Komputer dengan arsitektur 64-bit umumnya memerlukan memori yang lebih besar dibandingkan dengan komputer dengan arsitektur 32-bit. Dengan mengetahui arsitektur komputer, kebutuhan memori dari suatu program dapat diperkirakan.

Pada penilaian jawaban peserta, diperlukan pengukuran waktu dan memori yang sangat akurat. Pengukuran waktu atau memori yang tidak akurat akan menimbulkan ketidakadilan dalam melakukan penilaian jawaban peserta. Dengan hanya memerhatikan spesifikasi dari CPU dan sistem operasi pada komputer peserta, pengukuran waktu dan memori yang akurat sulit dilakukan. Program yang berjalan pada sistem operasi 64-bit tidak selalu memerlukan memori dua kali dari program yang berjalan pada sistem operasi 32-bit. Kebutuhan memori bergantung pada isi dari program yang dijalankan dan beberapa faktor lain. Kecepatan eksekusi dari suatu program pun bergantung dari beberapa faktor lain sehingga pengukuran waktu dengan cara ini sulit untuk dilakukan.

III.1.1.2 CPU *Benchmarking*

Benchmarking dapat dilakukan untuk mengukur kinerja CPU dengan menggunakan sebuah program *test*. Program *test* dapat dijalankan pada CPU untuk mengukur wak-

tu eksekusinya. Program *test* yang dijalankan pada komputer pengguna akan dicatat jumlah instruksi dan waktunya. Jumlah instruksi dan waktu yang dibutuhkan untuk menjalankan program *test* dapat digunakan untuk mengukur kinerja CPU. Batas waktu dari jawaban peserta kemudian ditentukan berdasarkan hasil eksekusi program *test* tersebut.

Benchmarking pada komputer peserta memiliki celah keamanan. Peserta bisa saja menjalankan program yang berat ketika proses *benchmarking* dilakukan. Hal ini akan menyebabkan waktu eksekusi menjadi lambat dan dapat meningkatkan batasan waktu dari jawaban peserta.

Selain adanya celah keamanan, terdapat kesulitan lain yang ditemukan ketika melakukan *benchmarking* yaitu pembuatan program *test*. Program *test* perlu dibuat sedemikian rupa sehingga dapat memberikan pengukuran waktu dan memori yang akurat. Untuk mengukur waktu eksekusi dan penggunaan memori dari program peserta dengan kompleksitas yang berbeda, diperlukan program *test* yang berbeda. Hal ini mengakibatkan perlunya membuat program *test* untuk setiap jenis soal yang ada pada kompetisi *competitive programming*. Pembuatan program *test* ini kemudian memberatkan juri karena juri perlu memikirkan program *test* yang cocok sehingga dapat memberikan pengukuran waktu dan memori yang akurat.

III.1.1.3 *Benchmarking* Dengan Solusi Juri

Teknik *benchmarking* sebenarnya efektif digunakan oleh *autograder* untuk melakukan pengukuran waktu. Meskipun begitu, teknik *benchmarking* tidak efisien dan memiliki celah keamanan jika digunakan. Juri perlu membuat sebuah program *test* untuk setiap soal yang diberikan kepada peserta. Hal ini memberatkan pekerjaan juri sehingga dinilai tidak efisien. Selain itu, peserta dapat menjalankan proses yang berat pada komputernya sehingga memengaruhi keakuratan teknik *benchmarking*.

Terdapat teknik yang dapat digunakan untuk meningkatkan efisiensi teknik *benchmarking*. Teknik *benchmarking* dapat menjadi lebih efisien dengan menghindari pembuatan program *test* dari setiap soal pada kompetisi *competitive programming*. Setiap soal dari kompetisi *competitive programming* memiliki solusi yang telah dibuat juri. Solusi dari juri tersebut dapat digunakan sebagai program *test*. Solusi *juri* memiliki kompleksitas yang sesuai dengan soal yang telah dibuat oleh juri sehingga dapat menjadi program *test* yang dapat digunakan untuk mengukur waktu eksekusi dan penggunaan memori dari program peserta secara akurat. Dengan menggunakan solusi juri sebagai program *test*, juri tidak perlu membuat program *test* khusus sehingga proses *benchmarking* dapat menjadi lebih efisien.

Selain masalah efisiensi, teknik *benchmarking* juga memiliki permasalahan lain yaitu bagaimana menghindari kecurangan peserta. Peserta bisa saja menjalankan proses yang berat pada komputernya sehingga mengganggu keakuratan proses *benchmarking*. Hal ini menyebabkan sistem *autograder* menganggap bahwa komputer peserta merupakan komputer yang lambat dan mengakibatkan jawaban peserta yang lambat bisa jadi dianggap jawaban yang benar.

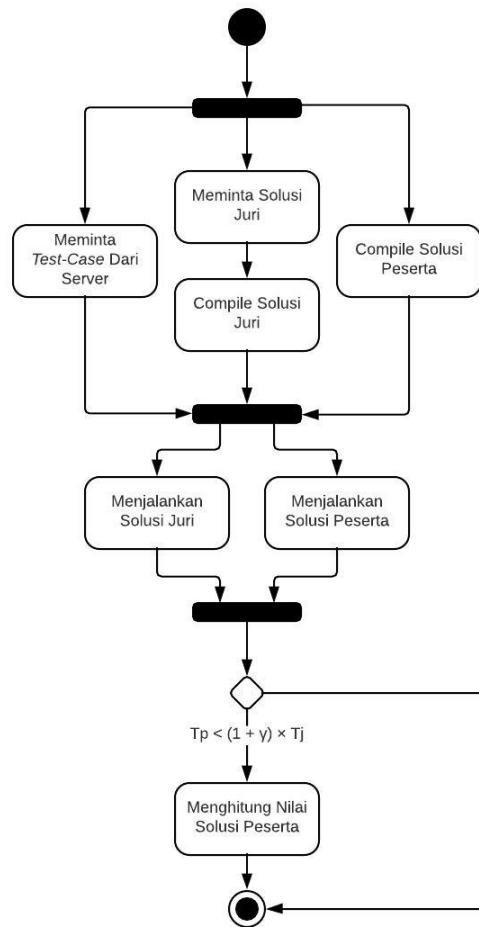
Terdapat teknik yang dapat digunakan untuk menghindari kecurangan peserta pada paragraf sebelumnya. Pengukuran waktu yang dilakukan pada saat *benchmarking* dapat dilakukan dengan hanya memperhitungkan *CPU Time*. *CPU Time* merupakan waktu dimana proses sedang dijadwalkan untuk dieksekusi oleh *CPU*. Ketika peserta menjalankan banyak proses yang berat, maka *CPU* akan dijadwalkan untuk mengeksekusi proses yang terkait dengan *autograder* secara lebih jarang. Hal ini dikarenakan sistem operasi membagikan *CPU* secara adil untuk seluruh proses yang ada. Dengan cara ini, waktu yang dijadwalkan untuk proses lain tidak akan dihitung dalam *benchmarking*.

Sistem operasi berbasis Linux menyediakan fitur *cgroup* yang dapat digunakan untuk melakukan pengukuran *CPU Time* dari suatu proses. Dengan menggunakan *cgroup*, *resource* yang digunakan oleh proses dapat dipantau. *Cgroup* memungkinkan suatu proses untuk memantau penggunaan CPU dan memori dari proses lain.

Autograder perlu menentukan apakah solusi peserta masih berada pada batas waktu yang diinginkan oleh juri. Hal tersebut dapat dicapai dengan membandingkan waktu eksekusi solusi peserta dengan waktu eksekusi solusi juri. Jika waktu eksekusi solusi peserta masih lebih cepat dari waktu eksekusi solusi juri, maka solusi peserta akan dianggap memenuhi batas waktu.

Meskipun solusi juri dan solusi peserta dieksekusi pada komputer yang sama. Waktu eksekusi kedua program tersebut dapat sedikit berbeda karena adanya perbedaan implementasi antara solusi peserta dan juri. Solusi peserta bisa saja lebih lambat atau lebih cepat dibandingkan dengan solusi juri. Untuk mengatasi perbedaan waktu ini, sebuah faktor toleransi dapat diterapkan pada sistem *autograder*. Solusi peserta dapat diterima jika waktu eksekusinya masih berada pada batas toleransi dari solusi juri. Secara matematis, solusi peserta akan diterima jika memenuhi persamaan berikut $T_{peserta} < (1 + \gamma) \times T_{juri}$, dengan γ adalah nilai dari faktor toleransi.

Dengan membandingkan penggunaan CPU dan memori dari solusi peserta dengan solusi juri, perhitungan waktu dan memori yang akurat dapat dilakukan. Meskipun peserta menjalankan program yang berat pada saat penilaian dilakukan, perbandingan waktu eksekusi dari program solusi juri dengan program solusi peserta akan tetap sama. Oleh karena itu, teknik ini digunakan untuk melakukan pengukuran waktu dan



Gambar III.1.2: Diagram Aktivitas Penilaian Solusi Peserta

memori dalam menyelesaikan tugas akhir ini. Gambar III.1.2 menjelaskan proses penilaian jawaban peserta dengan membandingkan dengan program solusi juri.

Kode. III.1: Contoh Program yang *IO bound*

```

#include <unistd.h>
int main() {
    while (1)
        sleep(1);
    return 0;
}

```

Dengan hanya menghitung *CPU Time* dari proses, terdapat masalah lain yang muncul. Peserta bisa saja mengirimkan jawaban berupa program yang bersifat *IO bound*. Program yang bersifat *IO bound* tidak menggunakan banyak *CPU Time*. Kode III.1 merupakan contoh program yang bersifat *IO bound*. Program tersebut tidak menggunakan banyak *CPU Time* karena hanya melakukan *sleep*. Pada saat *sleep*, proses yang

berjalan tidak akan dijadwalkan untuk dieksekusi oleh *CPU*. Hal tersebut menyebabkan proses dapat berjalan dengan sangat lama akan tetapi *CPU Time*-nya bernilai hampir nol. Program tersebut akan mengakibatkan *autograder* tidak pernah selesai melakukan pengukuran waktu karena *CPU Time*-nya sangat kecil dan proses tidak pernah selesai untuk dijalankan.

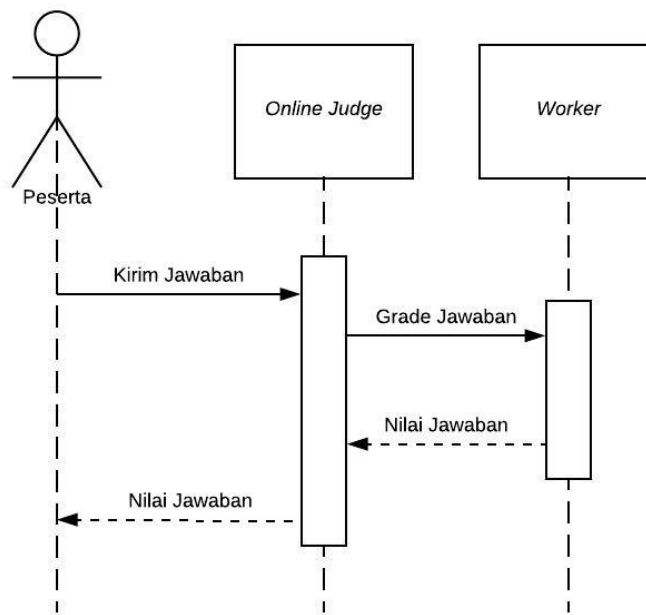
Untuk mengatasi proses yang bersifat *IO bound*, waktu eksekusi dapat dibatasi oleh *wall-clock time*. *Wall-clock* merupakan waktu sebenarnya dimana proses berada berada dalam sistem operasi komputer. Proses yang berjalan ketika melakukan *benchmarking* dapat dibatasi *Wall-clock time*-nya. Batas *wall-clock time* dapat diatur sedemikian rupa sehingga dapat dipastikan bahwa solusi peserta yang melewati batas ini dipastikan tidak lolos. Salah satu contoh batas *wall-clock time* adalah satu detik ditambah lamanya *CPU time* dari proses eksekusi solusi juri.

Penentuan batas *wall-clock time* sulit dilakukan pada komputer dengan spesifikasi yang berbeda-beda. Komputer yang berbeda mungkin memiliki kecepatan *IO* yang berbeda. Hal ini mengakibatkan juri perlu memerhatikan spesifikasi setiap komputer peserta untuk mengatur nilai *wall-clock time* yang tepat. Selain itu, setiap proses pada sistem operasi yang berbasis Linux memiliki prioritas yang dapat diatur. Peserta bisa saja menjadikan sistem *autograder* memiliki prioritas rendah sehingga proses yang melakukan pengukuran waktu jarang dijadwalkan oleh sistem operasi. Hal tersebut mengakibatkan berkurangnya keakuratan proses *benchmarking*.

Untuk mengatasi permasalahan pada paragraf di atas, kecepatan *IO* dari setiap komputer peserta yang digunakan untuk pengukuran waktu dapat dibatasi sehingga seluruh komputer peserta memiliki kecepatan *IO* yang sama. Selain itu, proses *autograder* dapat diatur untuk memiliki prioritas yang tinggi sehingga lebih sering dijadwalkan oleh sistem operasi. Pada tugas akhir ini solusi tersebut tidak digunakan karena masih perlu adanya penelitian lebih lanjut. Pada tugas akhir ini, ketika pengukuran waktu sudah melebihi batas *wall-clock time*, *autograder* akan memberikan *verdict Internal Error* yang menandakan proses penilaian gagal. Penilaian yang gagal tidak akan dihitung dan akan dijadwalkan untuk dikerjakan oleh komputer peserta lain.

III.1.2 Load Balancing

Dalam membangun sistem *autograder*, diperlukan pembagian kerja kepada *worker-worker* yang ada. *Worker* yang dimaksud adalah komputer yang bekerja melakukan penilaian jawaban peserta. Dalam tugas akhir ini, *worker* merupakan komputer peserta. Karena spesifikasi komputer tiap *worker* berbeda-beda, diperlukan teknik pembagian kerja yang adil kepada seluruh seluruh *worker*. Selain itu, aspek keamanan juga



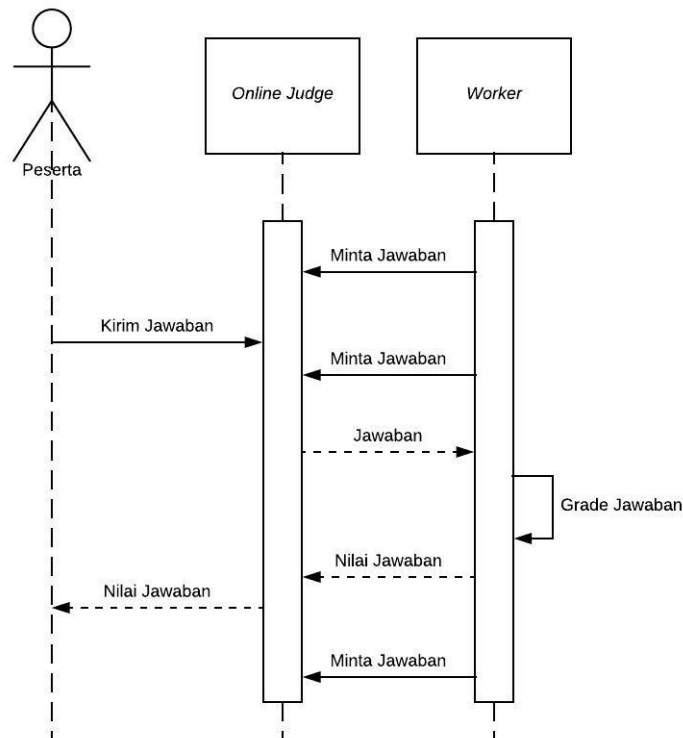
Gambar III.1.3: *Push-Based Load Balancer*

perlu diperhatikan karena jawaban peserta akan dikirimkan ke *worker* lain. Terdapat beberapa pendekatan pembagian kerja yang dapat digunakan, yaitu: *push-based*, *pull-based*, dan *self-grading*.

Pada pendekatan *push-based*, sistem *autograder* memerlukan sebuah *master* yang akan membagikan pekerjaan ke seluruh *worker*. *Master* akan menentukan *worker* mana yang akan diberikan suatu pekerjaan. Dengan menggunakan metode ini, *master* perlu mengetahui informasi *resource* dari seluruh *worker* yang ada. Setiap *worker* perlu mengirimkan informasi *resource*-nya kepada *master*. Metode ini cukup sulit untuk dilakukan karena perlunya mengimplementasikan algoritma *load balancing* pada *master*. Gambar III.1.3 menggambarkan alur pembagian kerja dengan metode *push-based*.

Pada pendekatan *pull-based*, diperlukan juga sebuah *master* yang akan menyimpan seluruh pekerjaan yang perlu diselesaikan. Pendekatan *pull-based* berbeda dengan pendekatan *push-based* dimana *master* lah yang memberikan pekerjaan kepada *worker*. Pada pendekatan *pull-based*, *worker* yang sedang tersedia akan meminta pekerjaan kepada *master*. Dengan pendekatan ini, pembagian pekerjaan akan menjadi rata dengan sendirinya. Metode ini lebih mudah untuk diimplementasikan dibanding dengan metode *push-bashed* karena tidak perlu membuat algoritma *load balancing* yang spesifik pada *master*. Gambar III.1.4 menggambarkan alur pembagian kerja dengan metode *pull-based*.

Selain pendekatan *push-based* dan *pull-based*, terdapat pendekatan lain yang lebih



Gambar III.1.4: *Pull-Based Load Balancer*

sederhana yaitu *self-grading*. Pada *self-grading*, peserta akan menilai jawabannya sendiri. Dengan menggunakan pendekatan ini, tidak diperlukan adanya *master*. Meskipun begitu, pendekatan *self-grading* memiliki celah keamanan karena peserta mengetahui jawaban siapa yang dinilai pada komputernya. Hal ini memungkinkan peserta untuk melakukan serangan-serangan yang mengakibatkan jawaban yang dinilai pada komputernya selalu dianggap benar.

Pendekatan *pull-based* lebih adil dan lebih mudah diimplementasikan dibandingkan dengan pendekatan *push-based*. Pendekatan *pull-based* lebih aman dibandingkan dengan pendekatan *self-grading* karena identitas jawaban yang dinilai pada *worker* lebih rahasia. Oleh karena itu, pendekatan *pull-based load balancing* digunakan pada tugas akhir ini karena lebih adil dan lebih aman dibandingkan dengan dua pendekatan lainnya.

III.1.3 Evaluasi Jawaban Menggunakan Sandbox

Dalam melakukan penilaian terhadap jawaban peserta, diperlukan adanya proses kompilasi dan eksekusi terhadap program yang dikirimkan peserta. Proses kompilasi dan eksekusi ini akan dijalankan pada *worker*. Proses ini dapat membahayakan lingkungan

worker jika peserta mengirimkan kode program yang berbahaya. Untuk menghindari kerusakan pada *worker*, proses ini perlu diisolasi sehingga tidak berpengaruh pada lingkungan *worker*. Teknik untuk mengisolasi proses eksekusi program ini dinamakan *sandboxing*. Terdapat beberapa teknik *sandboxing* yang dapat digunakan seperti *virtual machine* dan *containerization*.

III.1.3.1 Menggunakan *Virtual Machine*

Virtual machine memberikan isolasi kepada program pada tingkat *hardware*. Komputer yang ada pada saat ini memiliki *hypervisor* yang dapat digunakan untuk menyimulasikan *hardware* dari komputer. Dengan menggunakan *virtual machine*, komputer dapat menjalankan sistem operasi virtual di atas sistem operasi yang sedang berjalan. *Virtual machine* memberikan layanan isolasi yang sangat aman.

Meskipun keamanan *virtual machine* sangat tinggi, akan tetapi banyak *overhead* yang ditimbulkan. Untuk menyalakan *virtual machine*, membutuhkan waktu yang lama dan memori yang cukup besar. Selain itu, diperlukan adanya sistem operasi baru yang berjalan di atas *virtual machine* yang dibuat. Hal ini menyebabkan proses penilaian jawaban peserta menjadi lambat dan membutuhkan sangat banyak memori.

III.1.3.2 Menggunakan *Container*

Container merupakan suatu teknik untuk mengisolasi proses pada tingkat *software*. Dengan menggunakan *container*, proses yang berjalan pada sistem operasi berbasis Linux dapat dibatasi *resource*-nya. Penggunaan memori dan CPU dari suatu proses dapat dibatasi sehingga tidak membebani komputer peserta. Selain itu, Linux memiliki fitur yang memungkinkan pengguna mengisolasi suatu proses sehingga proses tersebut tidak dapat mengetahui informasi rahasia yang ada pada komputer peserta. *Container* dapat dibuat dengan memanfaatkan fitur dari sistem operasi berbasis Linux yaitu: *chroot*, *namespace*, *rlimit* dan *cgroup*. Pada tugas akhir ini, *container* digunakan untuk melakukan isolasi pada proses yang berjalan pada komputer peserta. Teknik ini dipilih karena memberikan isolasi yang cukup, dan tidak menimbulkan *overhead* yang sangat besar seperti teknik isolasi dengan *virtual machine*.

Chroot memungkinkan proses pada sistem operasi berbasis Linux untuk memiliki *file-system* sendiri. Hal ini dapat dilakukan dengan mengganti *root* dari proses ke *directory* tertentu. Dengan mengganti *root*, maka proses tidak dapat mengakses *file* yang berada di luar *directory root*. *Chroot* dapat dilakukan dengan melakukan pemanggilan *system call chroot*. *System call chroot* hanya dapat dilakukan oleh proses yang memiliki ak-

ses *root* atau memiliki *capability CAP_SYS_CHROOT*. Pada tugas akhir ini, program *autograder* diberikan akses *root* sehingga dapat melakukan pemanggilan *system call chroot*.

Namespace merupakan fitur dari Linux yang memungkinkan pengisolasian suatu proses sehingga tidak mengetahui informasi lain di luar *namespace*-nya. Setiap proses pada Linux memiliki *namespace*. Proses yang berada pada *namespace* tidak akan mengetahui informasi dari *namespace* lain. Hal ini memungkinkan suatu proses untuk diisolasi sehingga tidak mengetahui adanya proses lain, *user* lain, ataupun *network interface* lain. Dengan menggunakan *namespace*, keamanan dari lingkungan *worker* dapat dijaga karena jawaban peserta tidak mengetahui informasi mengenai komputer peserta. Pada tugas akhir ini, proses evaluasi jawaban peserta dilakukan di dalam *namespace* yang terpisah dari *namespace* pengguna.

Setiap proses pada sistem operasi berbasis Linux memiliki batas *resource* yang dapat digunakan. Batas tersebut dapat diatur dengan melakukan pemanggilan *system call setrlimit*. Proses yang menggunakan *resource* melebihi batas akan dihentikan oleh sistem operasi. Pada tugas akhir ini, *system call setrlimit* digunakan untuk membatasi jumlah proses yang dapat diciptakan, jumlah *open file descriptor* yang dapat dimiliki, dan ukuran file yang dapat diciptakan oleh program peserta.

Cgroup merupakan fitur dari sistem operasi berbasis Linux yang dapat digunakan untuk membatasi penggunaan *resource* suatu proses. Selain itu, *cgroup* juga dapat digunakan untuk memantau penggunaan *resource* dari proses tersebut. Pada tugas akhir ini, *cgroup* digunakan untuk mengukur penggunaan CPU dan memori dari proses yang berjalan dan menghentikan proses yang menggunakan *resource* secara berlebihan.

III.1.4 Kompilasi Program

Penilaian terhadap jawaban peserta memerlukan adanya tahap kompilasi agar program dapat dijalankan. Keamanan dari proses kompilasi tentunya juga perlu dijaga. Keamanan dari proses kompilasi dapat dicapai dengan menggunakan *sandbox* seperti yang sudah dijelaskan pada subbab sebelumnya.

Salah satu aspek yang perlu diperhatikan dalam melakukan kompilasi adalah pengadaan *compiler*. *Compiler* merupakan program yang digunakan untuk melakukan kompilasi kode sumber menjadi program yang dapat dijalankan pada sistem operasi. Komputer peserta tentunya perlu memiliki *compiler* untuk melakukan kompilasi. Untuk menjaga keadilan, seluruh *compiler* yang ada pada komputer peserta perlu memiliki versi yang sama. *Compiler* dengan versi yang berbeda dapat menghasilkan program

dengan kemampuan yang berbeda. Terkadang *compiler* dengan versi yang lebih baru dapat memberikan optimisasi pada program sehingga berjalan dengan lebih cepat. Perbedaan versi *compiler* pada komputer peserta akan memberikan ketidakadilan dalam proses penilaian.

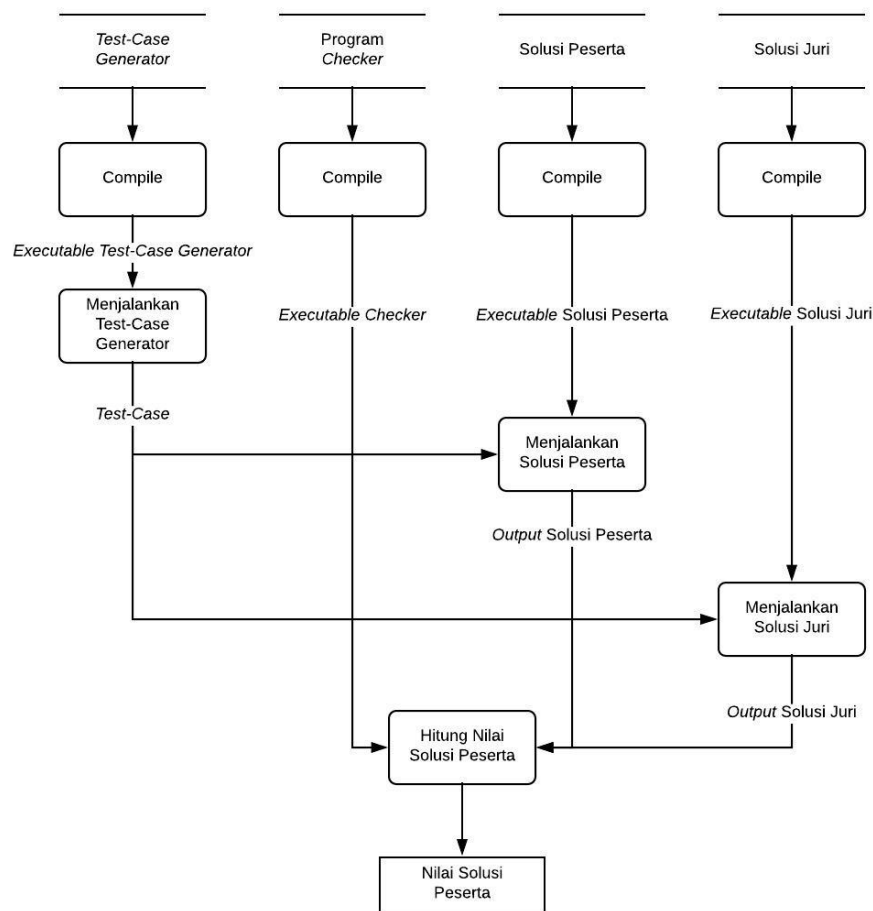
Masalah pada paragraf sebelumnya diatasi dengan melengkapi *autograder* dengan *compiler*. Karena ukuran *compiler* umumnya cukup besar, maka *compiler* di-compress ketika didistribusikan ke komputer peserta. Setiap sistem *autograder* dijalankan, *compiler* akan di-extract sehingga siap digunakan.

III.1.5 Pengiriman *Test-Case* Ke *Worker*

Dalam melakukan penilaian jawaban peserta perlu adanya pengiriman *test-case* dari *server* ke *worker*. *Test-case* merupakan informasi yang digunakan untuk menentukan kebenaran jawaban peserta sehingga informasi ini bersifat rahasia dan tidak boleh diketahui oleh peserta maupun orang lain di luar kompetisi. *Test-case* digunakan sebagai masukan pada saat menjalankan program solusi peserta dan juri. Kebenaran dari solusi peserta dinilai dari keluaran yang dihasilkan. Salah satu cara untuk menilai kebenaran program peserta tersebut adalah dengan membandingkan keluaran program solusi peserta dengan keluaran program solusi juri. Jika keluaran solusi peserta sama seperti keluaran solusi juri, maka solusi peserta akan dianggap benar. Akan tetapi, seringkali keluaran dari program solusi peserta tidak harus sama persis dengan solusi juri. Soal pada *competitive programming* seringkali memiliki lebih dari satu solusi yang sah dan tidak mungkin untuk menuliskan semua solusi satu per satu. Oleh karena itu, diperlukan juga adanya program *checker* yang digunakan untuk menentukan kebenaran solusi peserta. Program *checker* akan menilai kebenaran solusi peserta berdasarkan keluaran dari program solusi peserta dan program solusi juri.

Test-case dan *checker* merupakan informasi rahasia yang perlu dipertukarkan antara *worker* dan *server*. Pertukaran informasi rahasia ini dapat dilakukan dengan menggunakan teknik kriptografi dimana informasi akan dienkripsi terlebih dahulu sebelum dikirimkan. Proses enkripsi dan dekripsi dapat dilakukan pada lapisan *transport* menggunakan TCP dengan TLS.

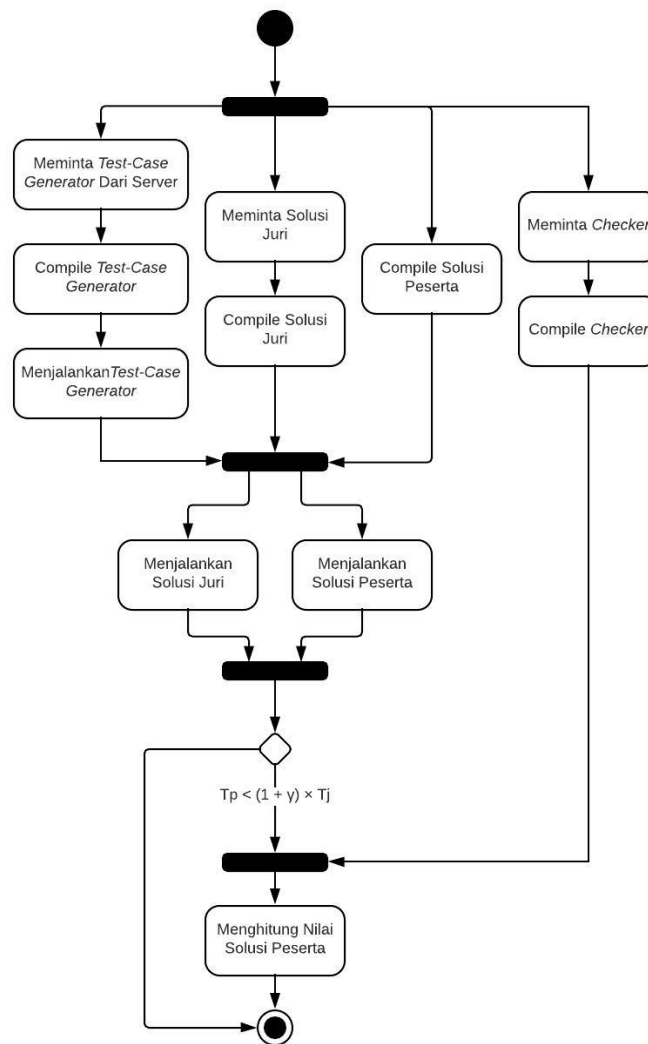
Dengan menggunakan TCP dan TLS, kebocoran informasi yang dikirimkan melalui jaringan dapat dihindari sehingga tidak ada pihak ketiga yang dapat mengetahui isi dari informasi tersebut. Akan tetapi, karena informasi ini diterima oleh komputer peserta, peserta dapat dengan mudah mengetahui isi dari informasi tersebut. Oleh karena itu, diperlukan satu lapisan enkripsi lagi pada aplikasi yang berjalan pada komputer peserta.



Gambar III.1.5: Diagram Aliran Data Pada Proses Penilaian Solusi Peserta

Test-case umumnya merupakan file *text* yang cukup besar. Pengiriman file ini dapat menghabiskan banyak *bandwidth*. Meskipun begitu, *test-case* biasanya dibuat dengan cara dibangkitkan oleh sebuah program yang dibuat oleh juri. *Test-case* yang sama dapat dibangkitkan berulang-kali dengan menjalankan program tersebut. Untuk mengurangi kebutuhan *bandwidth*, *test-case* tidak perlu dikirim, melainkan program pembangkit *test-case* saja yang dikirim. *Test-case* yang akan digunakan untuk melakukan penilaian solusi peserta dibangkitkan dengan program pembangkit *test-case* ini seperti pada Gambar III.1.5. Diagram pada Gambar III.1.6 menggambarkan alur proses penilaian jawaban peserta secara lengkap.

Pada tugas akhir ini, *test-case* dipertukarkan dengan mengirimkan program pembangkit *test-case* secara aman. Keamanan diperoleh dengan menggunakan dua kali enkripsi yaitu pada lapisan *transport* dan aplikasi. Terdapat beberapa protokol yang dapat digunakan dalam melakukan pengiriman informasi ini. Protokol yang dipilih pada tugas akhir ini adalah protokol HTTP. Protokol ini dipilih karena aman dan mudah untuk digunakan.

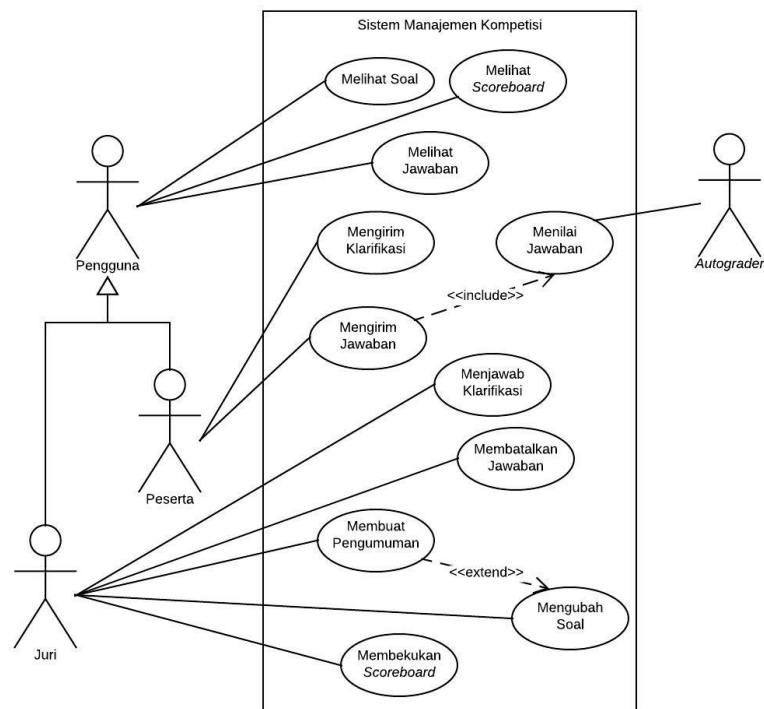


Gambar III.1.6: Alur Proses Penilaian Jawaban Peserta

III.2 Sistem Manajemen Kompetisi

Sistem manajemen kompetisi memberikan layanan kepada peserta kompetisi dan juri untuk berinteraksi. *Online judge* yang populer pada saat ini memberikan sistem manajemen kompetisi berbasis *web*. Sistem manajemen kompetisi ini dibutuhkan oleh peserta dan juri untuk melakukan aksi-aksi terkait kompetisi yang sedang berlangsung. Kebutuhan dari sistem manajemen kompetisi digambarkan oleh diagram *use case* pada Gambar III.2.1. Berikut ini adalah kebutuhan dari sistem manajemen kompetisi pada *online judge*:

1. Peserta dan juri dapat melihat soal.
2. Peserta dapat mengirim jawaban.

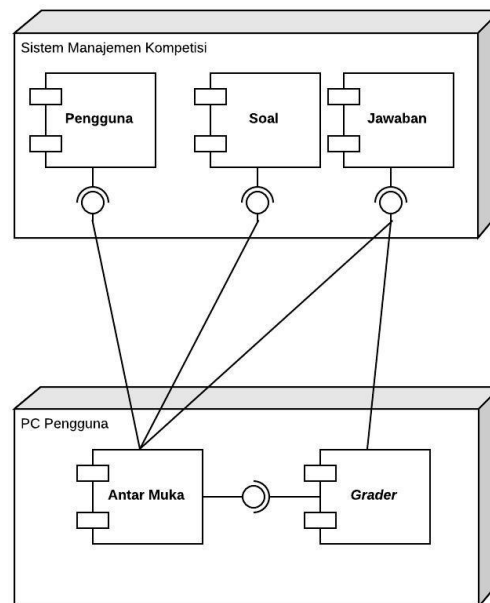


Gambar III.2.1: Diagram *Use Case* Dari Sistem Manajemen Kompetisi

3. Sistem *autograder* dapat menilai jawaban peserta.
4. Peserta dapat mengirim klarifikasi soal.
5. Juri dapat membuat pengumuman terkait kompetisi.
6. Juri dapat menjawab klarifikasi peserta.
7. Juri dapat mengubah soal.
8. Peserta dan juri dapat melihat *scoreboard*.
9. Juri dapat membekukan *scoreboard*.
10. Peserta dapat melihat jawaban yang dikirim olehnya.
11. Juri dapat melihat jawaban yang dikirim seluruh peserta.
12. Juri dapat mengatur jawaban peserta untuk tidak dinilai.

Pada tugas akhir ini, tidak semua kebutuhan dari sistem manajemen kompetisi diimplementasikan. Masalah yang ingin diselesaikan dalam tugas akhir ini tidak terletak pada sistem manajemen kompetisi, melainkan pada sistem *autograder*. Oleh karena itu hanya kebutuhan yang berkaitan dengan proses penilaian saja yang diimplementasikan pada tugas akhir ini. Kebutuhan sistem manajemen kompetisi yang diimplementasikan pada tugas akhir ini antara lain adalah:

1. Peserta dan juri dapat melihat soal.
2. Peserta dapat mengirim jawaban.
3. Sistem *autograder* dapat menilai jawaban peserta.
4. Juri dapat mengubah soal.
5. Peserta dapat melihat jawaban yang dikirim olehnya.
6. Juri dapat melihat jawaban yang dikirim seluruh peserta.



Gambar III.2.2: Diagram Komponen Sistem Manajemen Kompetisi

Sistem manajemen kompetisi yang digunakan pada tugas akhir ini mengikuti sistem manajemen kompetisi yang saat ini banyak digunakan. Sistem manajemen kompetisi ini dapat dibagi menjadi beberapa komponen utama yaitu: pengguna, soal, jawaban, antar muka dan *grader*. Komponen *grader* tidak lain adalah sistem *autograder* yang telah dipaparkan pada bagian III.1. Diagram komponen pada Gambar III.2.2 menggambarkan komponen yang digunakan dalam sistem manajemen kompetisi beserta keterhubungannya.

III.2.1 Komponen Pengguna

Kompetisi *competitive programming* ada yang bersifat tertutup dimana hanya peserta tertentu yang dapat mengikutinya dan terbuka dimana setiap orang dapat mengikutinya. Sistem *online judge* yang populer saat ini umumnya dapat menangani kedua jenis

kompetisi tersebut. Untuk memasuki kompetisi, peserta perlu membuat akun di sistem *online judge* tersebut terlebih dahulu. Peserta kemudian dapat memasuki kompetisi dengan cara melakukan *login* pada sistem tersebut. Untuk menangani pendaftaran peserta baru dan *login*, sistem *online judge* perlu memiliki komponen manajemen pengguna yang bertugas melakukan otentikasi dan otorisasi pengguna. Beberapa *online judge* memiliki komponen manajemen pengguna yang memanfaatkan layanan otentikasi dan otorisasi dari pihak ketiga seperti Google, Facebook, Github, dan Auth0. Terdapat juga sistem *online judge* yang melakukan otentikasi dan otorisasi tanpa menggunakan pihak ketiga, misalnya DomJudge dan Mooshak.

Komponen pengguna dibuat pada sistem manajemen kompetisi untuk melakukan otentikasi dan otorisasi pengguna. Untuk menyederhanakan persoalan, pada tugas akhir ini komponen pengguna yang bertugas melakukan otentikasi dan otorisasi pengguna tidak akan menggunakan layanan pihak ketiga. Beberapa *software development framework* seperti Django, Express, dan Laravel memiliki kemampuan untuk menangani masalah ini dengan mudah. Informasi pengguna akan disimpan pada sistem basis data yang sudah tersedia. Otentikasi pengguna akan dilakukan dengan sederhana, yaitu menggunakan *username* dan *password*.

Otorisasi pengguna juga dilakukan secara sederhana yaitu dengan sistem *Role-based access control*. Setiap pengguna akan diberikan *role* tertentu. Aksi-aksi yang dapat dilakukan oleh pengguna ditentukan oleh *role* yang dimilikinya. Setiap pengguna yang berhasil *login* kedalam sistem akan diberikan *token* unik. *Token* unik ini digunakan untuk menentukan *role* dari pengguna yang melakukan aksi pada sistem. *Token* yang akan digunakan dalam tugas akhir ini adalah *JWT (JSON Web Token)* karena mudah untuk digunakan dan aman.

III.2.2 Komponen Soal

Pada kompetisi *competitive programming*, deskripsi soal dapat diakses melalui sistem *online judge* yang biasanya berupa halaman web. Dalam beberapa kompetisi, peserta kompetisi akan mendapatkan *hard-copy* dari soal. Komponen soal dibutuhkan oleh sistem manajemen kompetisi untuk mengelola soal yang ada pada kompetisi. Komponen ini memberikan layanan kepada pengguna untuk melihat soal. Terkadang terdapat kesalahan pada soal ketika kompetisi berlangsung sehingga komponen ini perlu memberikan layanan kepada juri untuk mengganti soal.

Soal pada kompetisi *competitive programming* memiliki deskripsi yang merupakan penjelasan terhadap masalah yang harus diselesaikan peserta. Pada deskripsi soal terdapat penjelasan mengenai permasalahan yang dimaksud, format masukan, format ke-

luaran, contoh masukan, dan contoh keluaran. *Online judge* yang populer saat ini memberikan deskripsi soal kepada peserta dengan format *pdf* atau menampilkannya dalam halaman web. Komponen soal perlu menyimpan deskripsi soal untuk dapat memberikannya kepada pengguna. Deskripsi soal dapat disimpan pada *filesystem* dalam bentuk *pdf* atau disimpan dalam sistem manajemen basis data dalam bentuk *HTML*. Untuk memudahkan implementasi pada tugas akhir ini, deskripsi soal akan disimpan dalam basis data.

Setiap soal perlu memiliki program *test-case generator*, *solution*, dan *checker*. Ketiga buah program ini dibuat oleh juri untuk melakukan penilaian jawaban peserta. *Test-case generator* adalah program yang dibuat oleh juri untuk membangkitkan *test-case* seperti yang sudah dijelaskan pada bagian III.1.5. *Solution* adalah program yang merupakan solusi valid dari soal. Program solusi ini dibuat oleh juri untuk dibandingkan dengan jawaban peserta seperti yang sudah dijelaskan pada bagian III.1.1.3. Program *checker* akan digunakan oleh *worker* untuk menentukan kebenaran jawaban peserta. Terkadang terdapat lebih dari satu jawab yang benar pada sebuah soal. Oleh karena itu, program *checker* ini dibutuhkan untuk menilai jawaban peserta.

Program *test-case generator*, *solution*, dan *checker* dapat disimpan dalam *filesystem* atau sistem basis data. Untuk memudahkan implementasi pada tugas akhir ini, ketiga program ini akan disimpan dalam bentuk *source code* pada sistem basis data.

III.2.3 Komponen Jawaban

Sistem manajemen kompetisi perlu menyediakan layanan kepada peserta untuk mengirimkan jawaban. Jawaban yang dikirimkan peserta berupa program dalam bentuk *source code* dalam bahasa pemrograman yang diizinkan oleh juri. Selain itu, sistem manajemen kompetisi juga harus memberikan layanan kepada peserta untuk dapat melihat kembali jawabannya. Juri juga perlu mengawasi jawaban dari peserta untuk menghindari adanya kecurangan. Oleh karena itu, sistem manajemen kompetisi juga harus dapat memberikan layanan kepada juri untuk melihat seluruh jawaban yang pernah dikirimkan oleh peserta. Untuk memudahkan implementasi, pada tugas akhir ini jawaban peserta akan disimpan dalam bentuk *source code* pada sistem basis data.

BAB IV

PENGEMBANGAN DAN PENGUJIAN

Pada Tugas Akhir ini, perangkat lunak yang dikembangkan berupa sistem manajemen kompetisi, *autograder* dan program antar muka pengguna. Sistem manajemen kompetisi dideploy pada komputer yang disediakan oleh juri dan bertindak sebagai *server*. *Autograder* dan program antar muka akan didistribusikan kepada peserta dan juri untuk dijalankan pada komputernya. Secara keseluruhan, perangkat lunak yang dikembangkan dinamakan *UGrade*. Nama tersebut berasal dari kata *you* dan *grade*. Nama tersebut dipilih karena proses penilaian atau *grading* dilakukan oleh masing-masing peserta.

Terdapat tiga program utama yang dikembangkan pada *UGrade* yaitu: *UGServer*, *UGJob* dan *UGDesktop*. *UGServer* berfungsi sebagai sistem manajemen kompetisi yang bertindak sebagai server. *UGJob* berfungsi sebagai *worker* dari *autograder*. *UGDesktop* berfungsi sebagai antar muka antara pengguna dengan sistem *online judge*. Selain tiga program utama tersebut, terdapat program lain yang digunakan untuk membantu proses pengembangan. Untuk melakukan *sandboxing*, dikembangkan program yang bernama *UGSbox*. Selain itu, untuk memudahkan proses testing, dikembangkan program *UGCtl* yang dapat digunakan sebagai antar muka pengguna dalam bentuk *command-line*.

IV.1 *UGServer* (Sistem Manajemen Kompetisi)

UGServer merupakan sistem manajemen kompetisi yang berfungsi untuk mengatur keberjalanannya kompetisi. *UGServer* memberikan layanan kepada peserta dan juri untuk berinteraksi dengan kompetisi. Layanan yang diberikan oleh *UGServer* antara lain:

1. Otentikasi dan otorisasi pengguna.
2. Pembuatan dan akses kompetisi.
3. Pembuatan dan akses soal pada kompetisi.

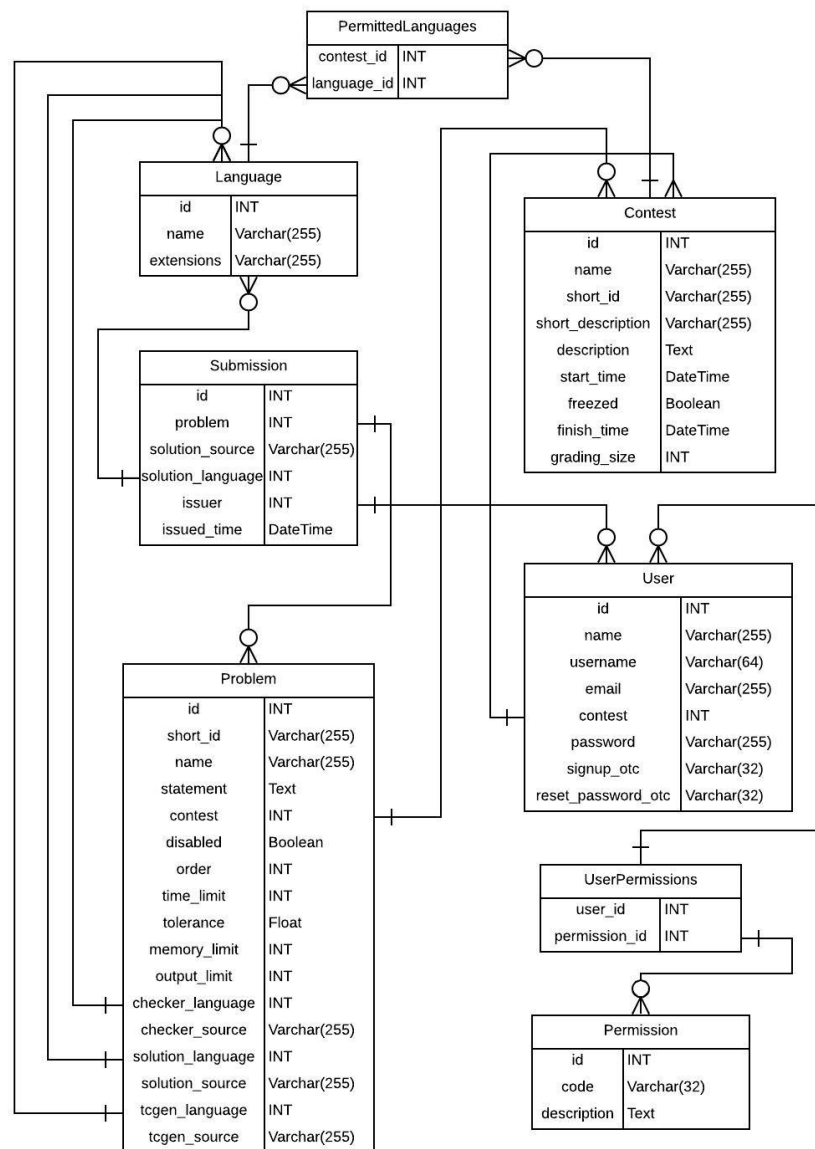
4. Pengiriman dan akses jawaban pada kompetisi.

Sebenarnya masih banyak layanan lain yang harus diberikan oleh sistem manajemen kompetisi. Akan tetapi, Tugas Akhir ini lebih difokuskan untuk mengembangkan *autograder* dibandingkan dengan sistem manajemen kompetisi. Hal ini dikarenakan tujuan dari Tugas Akhir ini lebih berfokus pada proses penilaian jawaban peserta. Oleh karena itu, fungsionalitas yang dipaparkan pada paragraf di atas sudah cukup untuk memenuhi tujuan dari Tugas Akhir ini.

UGServer dikembangkan dengan menggunakan *framework django*. Pengembangan perangkat lunak berbasis *web* dapat dengan cepat dan mudah dilakukan menggunakan *framework django*. Selain itu, *django* telah memberikan banyak fitur yang sangat membantu proses pengembangan perangkat lunak seperti otorisasi, otentikasi, dan *ORM*. Terdapat beberapa alternatif lain yang dapat meningkatkan kinerja sistem manajemen kompetisi seperti menggunakan *express* atau *GO*. Kedua alternatif tersebut tidak digunakan karena membutuhkan waktu pengembangan yang lama dan tidak terlalu memberikan perubahan yang signifikan. Oleh karena itu, *Framework django* dipilih untuk mengembangkan *UGServer* karena mudah dan cepat untuk dikembangkan.

UGServer memberikan *API* yang dapat digunakan oleh pengguna. *API* yang diberikan oleh *UGServer* berupa *GraphQL* yang berjalan di atas protokol *HTTP*. *GraphQL* digunakan karena mudah untuk diimplementasikan dan digunakan. *UGDesktop* dan *UGCtl* merupakan program antar muka pengguna yang menggunakan *API* ini. Terdapat beberapa alternatif lain yang dapat digunakan untuk mengembangkan *API* dari *UGServer* seperti menggunakan arsitektur *REST* atau menggunakan teknik seperti *RPC*. Alternatif tersebut tidak digunakan karena lebih tidak fleksibel dan lebih sulit untuk diimplementasikan.

Dalam menjalankan fungsinya, *UGServer* menggunakan sistem manajemen basis data untuk menyimpan informasi terkait kompetisi. Pada saat ini, *UGServer* dapat berjalan menggunakan sistem basis data *Postgresql* atau *SQLite*. *Framework django* memiliki fitur *ORM* yang siap digunakan untuk basis data yang bersifat relasional. Karena adanya fitur tersebut, data lebih mudah untuk diatur menggunakan basis data relasional. Oleh karena itu, pada Tugas Akhir ini penyimpanan data dilakukan menggunakan sistem basis data relasional. Gambar IV.1.1 merupakan skema basis data yang digunakan untuk menyimpan informasi terkait kompetisi.



Gambar IV.1.1: Skema Basis Data Sistem Manajemen Kompetisi

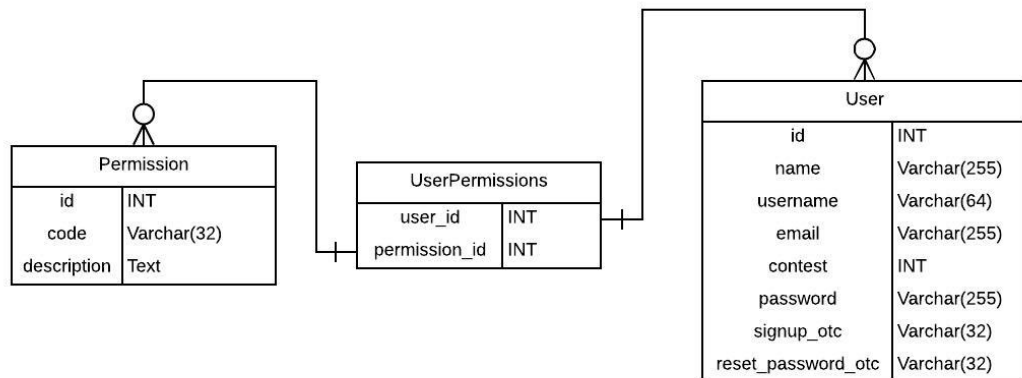
IV.1.1 Manajemen Pengguna

UGServer memberikan layanan untuk melakukan manajemen pengguna. Pengguna pada *UGrade* terikat pada suatu kompetisi. Pengguna pada suatu kompetisi tidak dapat mengikuti kompetisi lain. Pengguna yang ingin mengikuti dua buah kompetisi harus membuat dua akun. Pengguna dapat membuat akun dengan cara membuat kompetisi. Ketika seseorang membuat kompetisi, maka sebuah akun akan tercipta sesuai dengan alamat *email* orang tersebut. Selain itu, akun dapat dibuat dengan mengundang orang lain ke dalam kompetisi.

Akun yang pertama kali dibuat hanya mengandung informasi alamat *email* pengguna

dan kode untuk otentikasi. Kode tersebut akan dikirim ke *email* pengguna. Hal ini berfungsi sebagai cara untuk mengotentikasi pengguna dengan cara mengotentikasi *email* pengguna. Pengguna kemudian harus memasukkan kode tersebut untuk mengotentikasi dirinya. Pengguna yang telah terotentikasi akan memasukkan informasi mengenai nama lengkap, *username* dan kata sandi. Nama lengkap merupakan nama yang akan ditampilkan pada antar muka kompetisi. *Username* merupakan nama pengguna yang unik untuk setiap kontes dan bersifat mudah untuk dibaca oleh manusia. Setelah melakukan otentikasi *email*, pengguna selanjutnya dapat melakukan otentikasi dengan kombinasi kompetisi yang diikuti, alamat *email* dan kata sandi yang dipilih.

Setiap pengguna dalam suatu kompetisi memiliki alamat *email* dan *username* yang unik. Dalam dua kompetisi berbeda bisa saja terdapat akun dengan alamat *email* yang sama. Hal ini dikarenakan akun pengguna terikat pada suatu kompetisi sehingga seorang pengguna dapat diidentifikasi dengan menggunakan informasi kompetisi yang diikutinya dan alamat *email* atau *username*-nya.



Gambar IV.1.2: Skema Basis Data Sistem Manajemen Pengguna

Pengguna dalam perangkat lunak ini umumnya meliputi juri, peserta dan admin. Perangkat lunak *UGServer* sebenarnya tidak membedakan pengguna sebagai juri, peserta ataupun admin. Otorisasi dilakukan dengan memberikan label *permission* kepada setiap pengguna yang ada. Setiap pengguna memiliki himpunan *permission* yang menandakan aksi-aksi apa saja yang dapat dilakukan. Pada saat ini, terdapat beberapa *permission* yang mengatur hak akses pengguna yaitu:

1. *update:contest*: pengguna dengan *permission* ini dapat mengubah informasi kompetisi.
2. *create:problems*: pengguna dengan *permission* ini dapat membuat soal baru.
3. *read:problems*: *permission* ini menandakan seorang pengguna dapat melihat dan

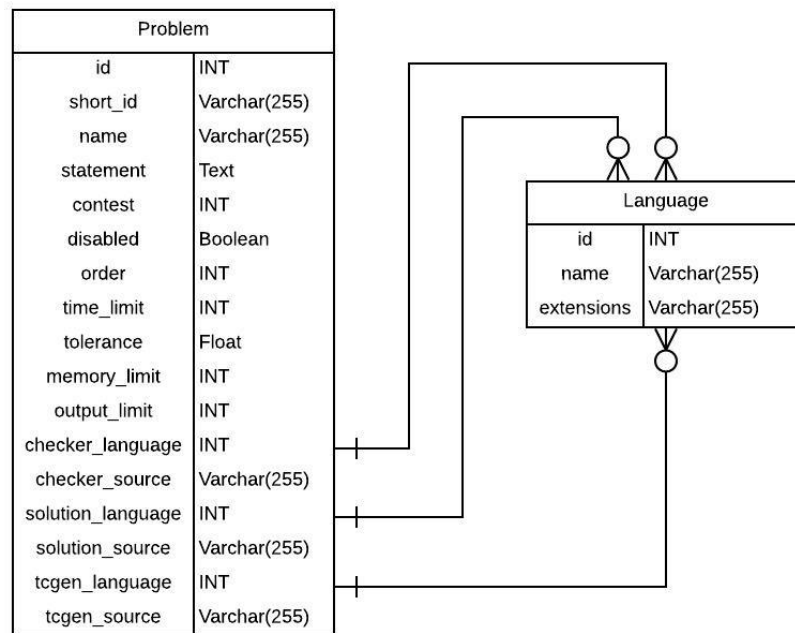
membaca soal.

4. *read:disabledProblems*: beberapa soal dapat ditandai sebagai *disabled* sehingga tidak dapat dibaca oleh pengguna yang tidak memiliki *permission* ini.
5. *update:problems*: pengguna dengan *permission* ini dapat mengubah informasi soal. Soal yang dapat diubah hanyalah soal yang dapat dilihatnya.
6. *delete:problems*: sebuah soal dapat dihapus oleh pengguna yang memiliki *permission* ini. Soal yang dapat diubah hanya soal yang dapat dilihat oleh pengguna yang bersangkutan.
7. *invite:users*: *permission* ini memungkinkan pengguna untuk mengundang pengguna lain.
8. *update:usersPermissions*: seorang pengguna dapat mengubah *permission* dari pengguna lain jika memiliki *permission* ini.
9. *delete:users*: pengguna dapat menghapus keanggotaan pengguna lain jika memiliki *permission* ini.
10. *read:submissions*: dengan *permission* ini, pengguna dapat melihat semua jawaban pengguna lain. *Permission* ini biasanya diberikan untuk juri.
11. *create:submissions*: *permission* ini memungkinkan pengguna untuk mengirimkan jawabannya. Pengguna tanpa *permission* ini dapat dipandang sebagai seorang penonton.

Pemberian *permission* untuk setiap pengguna bertujuan untuk meningkatkan *granularitas* pada hak akses untuk setiap pengguna. Gambar IV.1.2 merupakan skema basis data yang digunakan untuk melakukan manajemen pengguna.

IV.1.2 Manajemen Soal

Setiap soal pada *UGServer* akan terikat pada suatu kompetisi. Suatu soal tidak dapat berada pada dua kompetisi sekaligus. Jika terdapat soal yang sama pada kompetisi yang berbeda, maka soal tersebut harus ditambahkan dua kali dan tetap dianggap soal yang berbeda. Setiap soal pada kompetisi memiliki nama dan *short id*. Nama soal merupakan judul soal yang akan ditampilkan pada sistem antar muka yang digunakan pengguna. *Short id* merupakan kode soal yang unik untuk setiap kompetisi dan mudah untuk dibaca oleh manusia. *Short id* berguna untuk mengidentifikasi soal dengan mudah.



Gambar IV.1.3: Skema Basis Data Sistem Manajemen Soal

Informasi mengenai soal disimpan di sistem basis data pada tabel *problems*. Skema basis data yang digunakan untuk menyimpan informasi soal digambarkan pada Gambar IV.1.3. Aksi-aksi yang dapat dilakukan oleh pengguna terhadap soal ditentukan berdasarkan *permission* yang dimiliki oleh pengguna tersebut.

Setiap soal memiliki deskripsi soal. Deskripsi soal merupakan penjelasan mengenai soal, format masukan, format keluaran, contoh masukan dan contoh keluaran. Deskripsi soal ditulis dalam format *markdown* dan disimpan pada *field statement* pada tabel *problems*. Pemilihan format *markdown* bertujuan untuk kemudahan melakukan penyimpanan deskripsi soal. Beberapa sistem *online judge* lain memungkinkan melakukan penyimpanan soal dalam bentuk *file* seperti *pdf*. *UGServer* tidak mendukung penyisipan *file* pada deskripsi soal. Hal ini bertujuan untuk memudahkan implementasi.

Untuk melakukan penilaian terhadap jawaban peserta, setiap soal dilengkapi dengan beberapa informasi terkait penilaian. Beberapa informasi yang terkait dengan penilaian adalah: batas waktu eksekusi, batas penggunaan *memory*, batas *output file* yang dihasilkan, faktor toleransi, *testcase generator*, solusi juri dan *checker*. Batas waktu eksekusi disimpan pada *field timelimit* dalam bentuk bilangan bulat yang menyatakan lamanya batas waktu eksekusi dalam satuan milidetik. Batas waktu ini digunakan oleh *autograder* untuk menghentikan program secara paksa ketika berjalan terlalu lama. Batas penggunaan *memory* dan *output file* yang dihasilkan disimpan pada *field*

memorylimit dan *outputlimit* di tabel *problems*. Batas tersebut disimpan dalam bentuk bilangan bulat yang menyatakan batas dalam satuan *bytes*. Faktor toleransi disimpan pada *field tolerance* di tabel *problems* sebagai *float*.

Dalam melakukan penilaian, *autograder* memerlukan informasi mengenai *testcase generator*, solusi juri, dan *checker*. Informasi tersebut berupa *source code* pada bahasa pemrograman tertentu. *UGServer* menyimpan informasi bahasa pemrograman yang digunakan oleh *testcase generator*, solusi juri dan *checker* pada sistem basis data sebagai *foreign key* ke tabel *languages*. Informasi bahasa pemrograman tersebut disimpan pada *field tcgen_language*, *solution_language*, dan *checker_language* di tabel *problems*. Informasi mengenai kode sumber tidak disimpan dalam sistem basis data karena dianggap kurang efisien jika ukuran *source code* terlalu besar. Oleh karena itu, kode sumber disimpan sebagai *file* yang berada pada *disk*. Dengan begitu, sistem basis data hanya perlu menyimpan informasi mengenai alamat file dari *source code* yang disimpan.

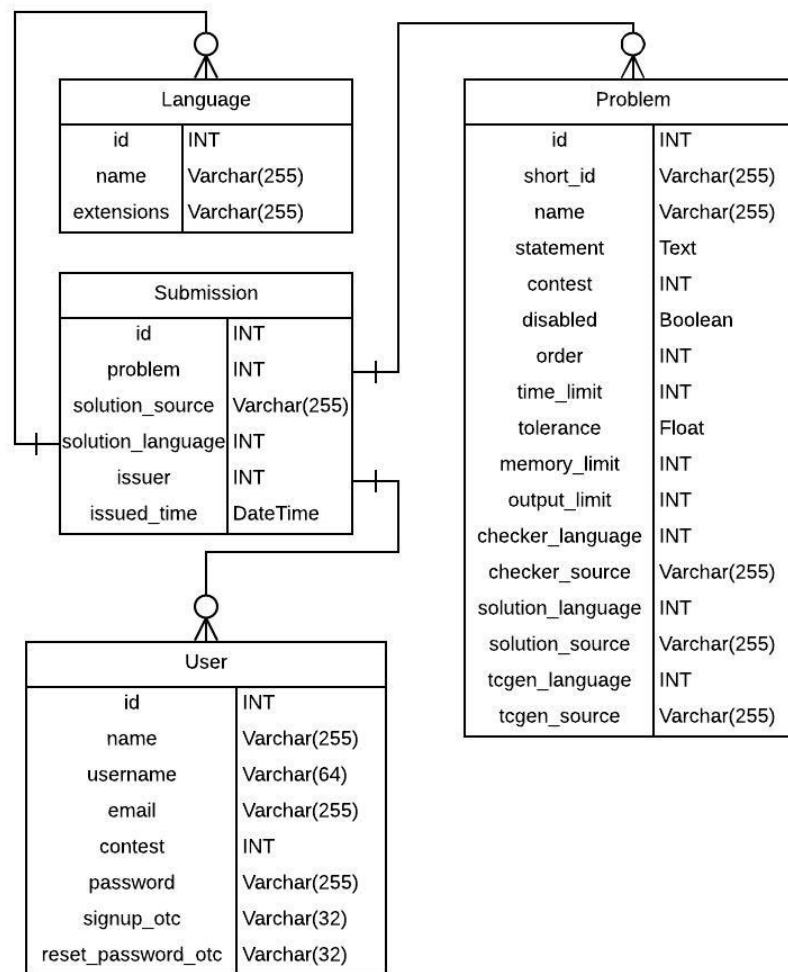
Pengguna dapat melakukan beberapa aksi pada soal yang ada pada sistem seperti: melihat daftar soal, membaca soal, menambah soal, mengubah soal, dan menghapus soal. Pengguna hanya dapat melakukan aksi-aksi tersebut sesuai izin yang dimilikinya seperti yang sudah dijelaskan pada bagian IV.1.1. Aksi-aksi tersebut dapat dilakukan menggunakan *GraphQL API*.

IV.1.3 Manajemen Jawaban

Peserta maupun juri dapat mengirimkan jawabannya terhadap suatu soal jika memiliki *permission create:submissions*. Jawaban terhadap suatu soal berupa *source code* yang ditulis pengguna dalam bahasa pemrograman tertentu yang diizinkan. Pada saat ini, bahasa pemrograman yang dapat digunakan adalah C dan C++. Pengguna dapat mengirimkan jawaban melalui *GraphQL API*.

Source code dari jawaban dari pengguna akan disimpan dalam bentuk *file* yang berada pada *disk*. Selain *source code*, terdapat informasi lain pada jawaban pengguna yang disimpan seperti bahasa pemrograman yang digunakan, waktu pengiriman jawaban, pengirim jawaban dan soal yang bersangkutan. Informasi tersebut disimpan dalam basis data pada tabel *submission*. Gambar IV.1.4 merupakan skema basis data yang digunakan untuk menyimpan jawaban.

Setiap jawaban yang berhasil dikirim oleh pengguna akan dinilai oleh *grader*. *Grader* akan berjalan secara otomatis ketika jawaban peserta disimpan. *Grader* kemudian akan menentukan *verdict* dari jawaban peserta. *Verdict* merupakan hasil penilaian terhadap jawaban peserta. Jenis *verdict* yang digunakan adalah:



Gambar IV.1.4: Skema Basis Data Sistem Manajemen Jawaban

1. *Accpeted*: menandakan bahwa jawaban memberikan keluaran yang benar dan memenuhi batasan.
2. *Wrong Answer*: menandakan bahwa jawaban memberikan keluaran yang salah tetapi memenuhi batasan.
3. *Memory Limit Exceeded*: menandakan bahwa jawaban menggunakan terlalu banyak *memory* melebihi batasan yang ditetapkan.
4. *Time Limit Exceeded*: menandakan bahwa jawaban melebihi batasan waktu yang sudah ditetapkan.
5. *Runtime Error*: menandakan bahwa jawaban tidak berhasil dieksekusi karena adanya kesalahan pada jawaban.
6. *Compile Error*: menandakan jawaban tidak dapat dikompilasi.

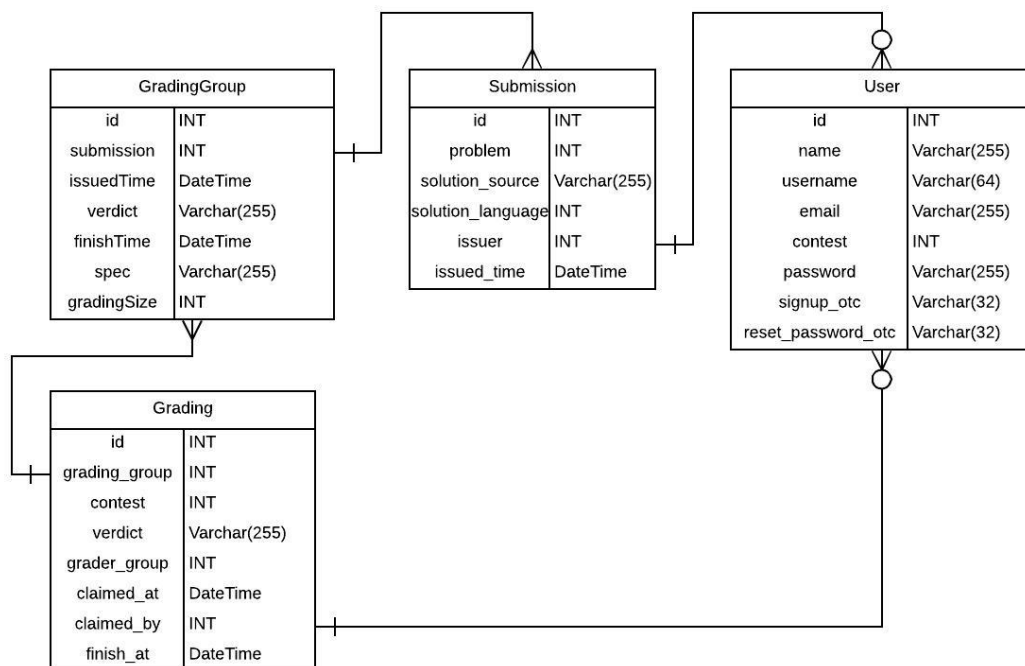
7. *Internal Error*: menandakan adanya kesalahan pada sistem *UGrade* sehingga jawaban tidak dapat dinilai.

IV.1.4 Penilaian Jawaban

Setiap jawaban yang dikirimkan oleh pengguna akan dinilai secara otomatis oleh *UGServer*. Penilaian dilakukan secara *asynchronous*. Oleh karena itu, peserta tidak perlu menunggu penilaian selesai untuk melakukan aksi-aksi lain. Seringkali terdapat beberapa masalah teknis yang mengakibatkan jawaban peserta perlu dinilai ulang. Oleh sebab itu, setiap jawaban peserta dapat dinilai lebih dari satu kali. Hasil penilaian yang digunakan adalah hasil penilaian yang terakhir.

IV.1.4.1 Grading Group

Penilaian pada sebuah jawaban disebut sebagai *grading group*. Setiap jawaban dapat dinilai lebih dari satu kali dan memiliki lebih dari satu *grading group*. Adanya *grading group* berguna untuk melakukan penilaian terhadap jawaban peserta lebih dari satu kali. Jika terdapat perubahan *testcase* atau perubahan batasan waktu pada suatu soal, maka perlu adanya penilaian ulang. Penilaian ulang dapat dilakukan dengan membuat *grading group* baru untuk setiap penilaian.



Gambar IV.1.5: Skema Penilaian Jawaban Pengguna

Informasi dari *grading group* disimpan dalam tabel *GradingGroups* pada sistem basis data. Setiap *grading group* memiliki informasi mengenai jawaban peserta, catatan waktu terciptanya *grading group*, *verdict*, spesifikasi penilaian, catatan waktu selesainya penilaian, dan *grading size*. Informasi-informasi tersebut disimpan sebagai *field* dalam tabel *GradingGroups*. Spesifikasi penilaian berupa *archive file* dan disimpan sebagai *file* dalam *disk*. Field *spec* pada tabel *GradingGroups* menyimpan informasi alamat *file* dari spesifikasi penilaian. Skema basis data yang digunakan untuk melakukan penilaian ditunjukkan pada Gambar IV.1.5.

Penilaian diawali dengan pembuatan spesifikasi penilaian. Spesifikasi penilaian berisi informasi mengenai *testcase generator*, solusi juri, *checker*, solusi peserta dan batasan soal. Informasi tersebut disimpan sebagai *file* dengan format *tar*. Spesifikasi penilain tersebut terdiri dari beberapa file sebagai berikut:

1. *tcgen.cpp*
merupakan kode sumber *testcase generator* dari soal. Ekstensi dari *file* dapat berbeda sesuai bahasa pemrograman yang digunakan.
2. *solution.cpp*
merupakan kode sumber solusi juri pada soal yang bersangkutan. Ekstensi dari *file* dapat berbeda sesuai bahasa pemrograman yang digunakan.
3. *checker.cpp*
merupakan kode sumber *checker* dari soal. Ekstensi dari *file* dapat berbeda sesuai dengan bahasa pemrograman yang digunakan.
4. *submission.cpp*
merupakan kode sumber solusi peserta. Ekstensi dari *file* dapat berbeda sesuai dengan bahasa pemrograman yang digunakan.
5. *lang.json*
file ini berisi *JSON document* yang menjelaskan bahasa pemrograman yang digunakan sebagai *testcase generator*, solusi juri, *checker* dan solusi peserta. *JSON document* pada *file* ini hanya berisi empat *key* yaitu *tcgen*, *solution*, *checker*, dan *submission*. Setiap *key* pada *file* ini memiliki *value* yaitu bahasa ID dari bahasa pemrograman yang digunakan.
6. *problem.json*
file ini berisi *JSON document* yang menjelaskan batasan dari soal. *JSON document* pada *file* ini berisi empat buah *key* yaitu *timeLimit*, *outputLimit*, *memoryLimit*, dan *tolerance*. Setiap *key* pada *file* ini memiliki *value* berupa bilangan yang menjelaskan batasan dari soal.

File spesifikasi penilaian ini nantinya akan diunduh oleh *autograder* yang berjalan pada komputer pengguna.

IV.1.4.2 *Grading Size*

Setiap penilaian yang ditandai dengan *grading group* memiliki suatu nilai *grading size*. Nilai dari *grading size* digunakan untuk mengurangi risiko serangan yang mungkin dilakukan pengguna. Sebuah penilaian dapat dilakukan oleh lebih dari satu *worker*. Setiap *worker* akan memberikan hasil penilaiannya kepada *UGServer*. Jawaban peserta akan dinilai benar ketika mayoritas dari *worker* yang melakukan penilaian memberikan *verdict accepted*. Nilai dari *grading size* mengindikasikan banyaknya *worker* yang melakukan penilaian.

Nilai *grading size* yang besar akan lebih aman karena dapat lebih mengurangi risiko kecurangan peserta. Hal ini dikarenakan kemungkinan terjadinya kecurangan lebih kecil sebab *worker* yang bekerja lebih banyak. Meskipun begitu, tingginya nilai *grading size* menyebabkan proses penilaian menjadi lebih lambat karena penilaian harus menunggu banyak *worker* untuk selesai terlebih dahulu. Sebaliknya, nilai *grading size* yang kecil dapat mempercepat penilaian akan tetapi mengurangi keamanan. Nilai *grading size* dapat diatur dalam pengaturan kompetisi oleh pengguna yang memiliki *permission update:contests*.

Karena setiap jawaban dapat dinilai lebih dari satu kali, perlu ada entitas yang merepresentasikan penilaian sebuah jawaban oleh sebuah *worker*. Entitas tersebut dinamakan *grading*. Sebuah *grading* memiliki informasi mengenai *grading group* terkait, *verdict*, *grader group*, catatan waktu penilaian, dan pengguna yang melakukan penilaian. Informasi ini disimpan di dalam basis data pada tabel *Gradings*.

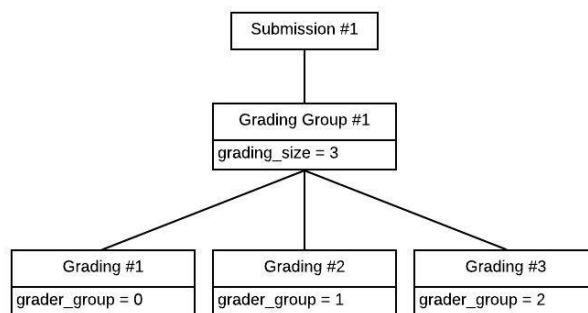
Secara singkat, setiap jawaban dapat memiliki beberapa *grading group* yang merepresentasikan sebuah penilaian terhadap jawaban. Hasil penilaian jawaban pada sebuah *grading group* merupakan hasil penilaian yang dilakukan oleh banyak *worker*. Banyaknya *worker* yang melakukan penilaian adalah sebanyak *grading size*. Setiap penilaian yang dilakukan oleh *worker* direpresentasikan menjadi sebuah *grading*. Secara singkat dapat dikatakan bahwa setiap *grading group* memiliki *grading* sebanyak *grading size*.

IV.1.4.3 *Grader Group*

Adanya nilai dari *grading size* bertujuan agar setiap jawaban dapat dinilai lebih dari satu *worker*. *Worker* yang melakukan penilaian terhadap suatu jawaban tentunya harus

merupakan *worker* yang berbeda. Untuk mengatasi hal ini, setiap *grading* dan *worker* memiliki sebuah nilai *grader group*. *Worker* dengan nilai *grader group* tertentu hanya dapat melakukan penilaian terhadap *grading* yang memiliki nilai *grader group* yang sama. Setiap penilaian jawaban peserta, akan dibuat *grading size* buah *grading* yang memiliki nilai *grader group* yang unik yaitu dari nol hingga *grading size* – 1. Setiap *worker* dengan *grader group* *K* hanya akan melakukan penilaian terhadap *grading* yang memiliki nilai *grader group* *K*.

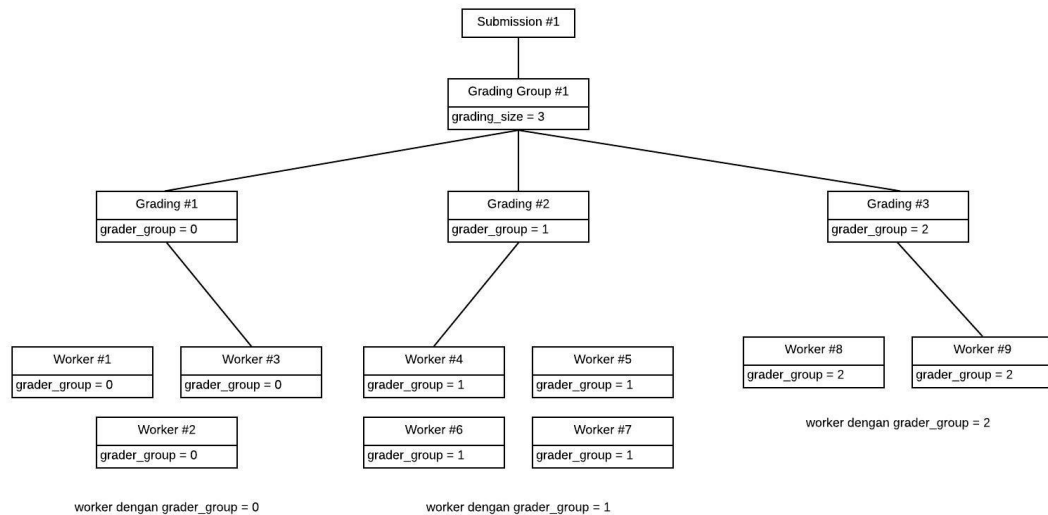
Nilai *grader group* dari setiap *worker* dihitung dengan cara melakukan *hash* pada ID *worker*. Setiap *worker* memiliki ID berupa integer yang unik. Nilai *grader group* sebuah *worker* dihitung dengan formula $grader_group = hash(W_{id}) \bmod grading_size$. Dengan mengasumsikan jumlah *worker* cukup banyak dibandingkan dengan nilai *grading size*, maka cara ini akan mengelompokkan menjadi *grading size* buah kelompok dengan anggota yang kira-kira sama banyak. Cara ini dipilih karena perhitungannya yang cepat, mudah dan dapat mengelompokkan *worker* secara rata.



Gambar IV.1.6: Contoh Proses Pembuatan *Grading*

Sebagai contoh, seorang pengguna telah melakukan pengiriman jawaban terhadap suatu soal. Setelah pengiriman jawaban tersebut berhasil dilakukan, akan tercipta sebuah *submission* yang dapat kita beri nama *Submission #1*. *UGServer* kemudian akan secara otomatis membuat *grading group* untuk menilai jawaban tersebut. Kita dapat memisalkan *grading group* tersebut bernama *Grading Group #1*. Pada *Grading Group #1* terdapat informasi mengenai *grading size*. Informasi *grading size* didapatkan dari konfigurasi kompetisi. Pengguna dengan *permission update:contest* dapat mengubah nilai ini. Dalam contoh ini, kita misalnya nilai *grading size* dari kompetisi tersebut adalah tiga. Setelah *grading group* terbentuk, *UGServer* akan membuat *grading size* buah *grading*. Dalam contoh ini berarti akan tercipta tiga buah *grading*. Untuk kemudahan penjelasan, tiga buah *grading* ini dapat kita sebut dengan *Grading #1*, *Grading #2*, dan *Grading #3*. Setiap *grading* yang diciptakan akan memiliki nilai *grader group* yang unik dan terurut dari nol hingga *grading size* – 1. Dalam contoh ini *Grading #1* memi-

liki nilai *grader group* nol, *Grading #2* memiliki nilai *grader group* satu, dan *Grading #3* memiliki nilai *grader group* dua. Gambar IV.1.6 menggambarkan *submission*, *grading group* dan *grading* pada contoh di paragraf ini.



Gambar IV.1.7: Contoh Proses Penilaian Oleh *Worker*

Setelah terbentuk tiga buah *grading* seperti pada paragraf sebelumnya, penilaian dapat mulai dilakukan oleh *worker*. *Worker* secara periodik akan selalu bertanya pada *UGServer* apakah terdapat *grading* yang perlu dikerjakan. *UGServer* akan memberikan *grading* yang sesuai dengan nilai dari *grader group worker* yang bersangkutan. Setiap *grading* memiliki informasi *claimed_by* yang akan berisi ID dari peserta yang sedang melakukan penilaian terhadap *grading* tersebut. Nilai tersebut bertujuan agar *grading* yang sedang dinilai tidak akan dinilai oleh *worker* lain. Gambar IV.1.7 menjelaskan pemberian *grading* pada *worker* yang dilakukan oleh *UGServer*.

Worker yang telah melakukan penilaian akan mengirimkan hasil penilaiannya kepada *UGServer*. *UGServer* kemudian akan mengubah nilai *verdict* dari *grading* dan mencatat waktu selesainya penilaian. Jika jumlah *grading* yang selesai dinilai sudah lebih dari setengah dari *grading size*, maka *verdict* dari *grading group* akan diubah. Setelah tahap tersebut, pengguna dapat melihat hasil akhir penilaiannya.

IV.2 *UGSbox*

Dalam menjalankan penilaian, *UGSbox* berperan dalam memberikan isolasi terhadap program-program yang berjalan di komputer pengguna. Program-program yang berjalan pada saat penilaian antara lain adalah: *compiler*, *testcase generator*, solusi peserta,

solusi juri, dan *checker*. Program-program ini dapat berisi kode yang membahayakan komputer pengguna yang menjalankannya. Oleh karena itu, diperlukan adanya sistem yang dapat mengisolasi program tersebut sehingga tidak membahayakan komputer pengguna. Pada Tugas Akhir ini, *UGSbox* berperan dalam memberikan isolasi tersebut.

UGSbox memberikan isolasi pada proses yang berjalan dengan menggunakan beberapa fitur dari linux. Isolasi yang diberikan *ugsbox* antara lain adalah sebagai berikut:

- isolasi *filesystem*.

Isolasi terhadap *filesystem* dilakukan dengan menggunakan *system call chroot*. Dengan menggunakan *chroot*, direktori *root* dari suatu proses dapat diganti menjadi direktori lain. Mengganti *root* dari proses mengakibatkan proses tersebut tidak dapat mengakses *file* yang ada di luar *root*. *Filesystem* yang akan digunakan oleh proses dapat dimuat dari *file* yang memiliki format *.tar.xz*. File tersebut dinamakan *image file*. Proses isolasi *filesystem* dimulai dengan melakukan ekstraksi *image file* kemudian mengganti *root* dari proses ke dalam direktori *image file* yang sudah berhasil diekstrak.

- *filesystem binding*.

UGSbox memberikan fitur *filesystem binding* yang dapat digunakan oleh pengguna untuk memasukkan *filesystem* lain ke dalam *filesystem* yang sudah diisolasi. Fitur ini berguna untuk menyimpan *testcase* ke dalam suatu *filesystem* khusus yang kemudian dapat dimuat ke dalam *filesystem* yang sudah diisolasi.

- isolasi *network*.

Untuk menjaga keamanan, setiap proses yang berjalan di dalam lingkungan yang diisolasi tidak boleh menggunakan *network* untuk berkomunikasi dengan dunia luar. Untuk mengatasi masalah ini, *UGSbox* menggunakan fitur *namespace* dari linux. Proses yang diisolasi akan mendapat *namespace* dengan *network interface* yang terisolasi dari *network interface* milik *host*.

- isolasi *user*.

Selain isolasi *network*, *namespace* pada Linux juga digunakan untuk melakukan isolasi *user*. Dengan mengisolasi *user*, proses yang terisolasi tidak dapat mengetahui adanya *user* lain di luar lingkungannya.

- batas penggunaan CPU dan *memory*.

Penggunaan CPU dan *memory* dari proses yang diisolasi tentunya perlu dibatasi. Hal ini bertujuan agar proses yang berjalan tidak membebani komputer pengguna ketika penilaian sedang berlangsung. Untuk mengatasi masalah ini, *UGSbox* menggunakan fitur *cgroup* dari linux. Dengan menggunakan *cgroup*,

penggunaan *memory* dan CPU dapat dibatasi dan dimonitor. Proses yang menggunakan CPU atau *memory* melebihi batasan akan dihentikan secara paksa.

- batas ukuran *file* yang dapat dihasilkan.

Program yang dikirimkan oleh pengguna mungkin saja memuat kode yang dapat memberatkan *disk* karena melakukan penulisan secara terus menerus. Hal ini dapat mengakibatkan komputer pengguna menjadi lambat karena *disk*-nya tidak dapat digunakan. Untuk mengatasi masalah ini, *UGSbox* membatasi ukuran *file* yang dapat diciptakan oleh proses di dalam lingkungan yang diisolasi. Pembatasan ini dicapai dengan menggunakan *system call setrlimit*.

- batas jumlah *file* yang dapat dibuka.

Untuk meningkatkan keamanan, jumlah *file* yang dapat dibuka oleh proses yang diisolasi juga dibatasi menggunakan *system call setrlimit*.

- batas proses yang dapat diciptakan.

Selain membatasi ukuran *file* yang dihasilkan, dan jumlah *file* yang dapat dibuka, *UGSbox* juga menggunakan *setrlimit* untuk membatasi proses yang dapat diciptakan.

Kode. IV.1: Contoh Hasil Eksekusi Perintah *ugsbox*

```
$ ugsbox guard --help
Usage:
  ugsbox guard [flags]

Flags:
  -b, --bind strings          bind host directory to sandbox
                              directory with format <hostdir>:<sandboxdir>. Warning: file
                              owner of binded directory will be changed
  -f, --file-size uint        generated file size limit
  -h, --help                  help for guard
  -i, --image string          compressed sandbox image (in .
                              tar.xz) path
  -m, --memory-limit uint     memory limit in bytes (default
                              67108864)
  -M, --memory-throttle uint  memory throttle in bytes (
                              default 268435456)
  -n, --nproc uint            limit process creation e.g.:
                              fork/exec
  -o, --open-file uint        open file limit
  -s, --stack-size uint       limit stack size in bytes
  -E, --stderr string         path (relative to sandbox) to
                              file to be used as stderr
```

```

-I, --stdin string          path (relative to sandbox) to
                             file to be used as stdin
-O, --stdout string         path (relative to sandbox) to
                             file to be used as stdout
-t, --time-limit uint       time limit in milisecond (
                             default 10000)
-T, --walltime-limit uint   wall clock time limit in
                             milisecond (default 10000)
-w, --working-directory string working directory of process (
                             default "/home")

```

Global Flags:

```

--debug    show debug log
--trace    show trace log

```

UGSBox dapat digunakan dengan dua cara yaitu: sebagai *executable program* atau sebagai *library*. Pengguna dapat menggunakan perintah *ugsbox guard* untuk menjalankan *UGSbox* melalui CLI. Kode IV.1 merupakan contoh hasil pemanggilan *ugsbox guard*.

Selain melalui CLI, *UGSbox* juga dapat digunakan melalui pemanggilan fungsi pada bahasa pemrograman GO. Untuk dapat menggunakan *UGSbox* sebagai *library*, pengguna dapat meng-*import package* <https://github.com/jauhararifin/ugrade> dan <https://github.com/jauhararifin/ugrade/sandbox>. Penggunaan *library* tersebut dapat dilihat pada dokumentasi *UGrade* di <https://godoc.org/github.com/jauhararifin/ugrade>.

IV.3 UGJob

UGJob merupakan implementasi dari *worker* pada keluarga perangkat lunak *UGrade*. *UGJob* dikembangkan menggunakan bahasa GO dan berperan dalam menilai jawaban pengguna. *UGJob* dikembangkan sebagai sebuah *executable* yang dapat dijalankan melalui CLI. *UGDesktop* akan secara periodik menjalankan *UGJob* untuk melakukan penilaian.

Setiap *grading* yang dihasilkan oleh *UGServer* akan dinilai oleh *worker* yang ada pada komputer pengguna. *Worker* pada komputer pengguna tidak perlu mengetahui adanya konsep *grading group*, *grading size*, dan *grader group*. *Worker* hanya perlu meminta *grading* pada *UGServer* untuk dinilai. Dalam meminta *grading*, *worker* hanya perlu mengirimkan ID dari pengguna yang bertindak sebagai *worker*. *UGServer* akan memberikan *grading* yang sesuai dengan ID pengguna dari *worker* tersebut.

Di sisi *worker*, semua *grading* yang perlu dinilai akan dipandang sebagai sebuah *job*.

Pada Tugas Akhir ini, *job* dapat disamakan dengan *grading* karena sebuah *grading* akan dianggap sebagai sebuah *job* oleh *worker*. Istilah *job* digunakan untuk meningkatkan abstraksi dari *worker*. *Worker* tidak perlu mengetahui konsep penilaian yang dilakukan di sisi *UGServer*. *Worker* tidak perlu mengetahui adanya konsep *grading* pada *UGServer*. Semua penilaian yang perlu dilakukan oleh *worker* akan dianggap sebagai sebuah *job*. Dengan menggunakan istilah *job*, sistem penilaian yang ada pada sisi *UGServer* dapat dengan mudah dimodifikasi tanpa memengaruhi *worker*.

Kode. IV.2: Contoh Hasil Eksekusi Perintah *ugjob consume*

```
$ ugjob consume --help
This program fetch job from server, execute it and send the
    result back to server.

Usage:
    ugjob consume [flags]

Flags:
    -h, --help                help for consume
    -u, --server-url string    Server url (default "http://localhost
                               :8000")
    -t, --token string         Your session token

Global Flags:
    --debug    show debug message
    --trace    show trace message
```

UGJob dapat dijalankan melalui CLI dengan mengetikkan perintah *ugjob consume*. *UGJob* memerlukan *token* yang bisa didapatkan oleh pengguna pada saat melakukan *sign in*. *Token* ini berguna untuk mengidentifikasi ID dari pengguna. *UGServer* akan menggunakan *token* untuk menentukan *job* mana yang akan dikerjakan oleh *worker*. Kode IV.2 merupakan contoh hasil pemanggilan *ugjob consume*.

Pemanggilan *ugjob consume* akan menjalankan *worker* pada komputer pengguna dan proses penilaian akan mulai dilakukan. Penilaian diawali dengan pengambilan *job* oleh *UGJob*. *UGJob* akan meminta *job* kepada *UGServer* dengan mengirimkan *GET HTTP request* ke *UGServer* dengan menyertakan *token* yang dimilikinya. *HTTP Request* tersebut akan dikirimkan ke URL */gradings* pada *UGServer* dengan menyertakan *token* pada bagian *header*.

Setiap permintaan *job* yang dilakukan oleh *UGJob* akan diproses oleh *UGServer*. *UGServer* akan memberikan *job* yang sesuai kepada *UGJob*. Pemberian *job* dilakukan dengan memandang *token* yang disertakan oleh *UGJob* ketika melakukan permintaan

job. Jika tidak terdapat *job* yang perlu dikerjakan, maka *UGServer* akan memberikan *HTTP response* dengan status 404. *UGServer* akan memberikan *job* kepada *UGJob* dengan bentuk *tar file* yang dikirimkan melalui *body* pada *HTTP response*. Selain itu, *UGServer* juga menyertakan *job token* pada setiap *job* yang dikirimkan. *Job token* ini berguna untuk membedakan antara satu buah *job* dengan yang lainnya.

Setelah mendapatkan *job* dari *UGServer*, *UGJob* akan melakukan penilaian dengan mengekstrak *job* yang didatarkannya. *Job* akan diekstrak pada suatu direktori sementara di */tmp*. Direktori */tmp* dipilih karena sistem secara otomatis akan menghapus isi dari direktori ini ketika *booting*. Selain itu, direktori ini juga sudah standar digunakan sebagai tempat penyimpanan *file* yang bersifat sementara. Direktori *Job* yang sudah berhasil dinilai secara otomatis akan dihapus oleh *UGJob*.

IV.3.1 Pembangkitan *Testcase*

Untuk melakukan penilaian, diperlukan adanya *testcase*. *Testcase* akan dibangkitkan dengan menggunakan program *testcase generator* yang disertakan pada *file job*. *Testcase generator* ini pertama-tama dikompilasi menjadi *executable file* kemudian dijalankan. *Testcase* yang dihasilkan oleh *testcase generator* disimpan pada direktori sementara di dalam */tmp*.

Program *testcase generator* hanya membangkitkan masukan dari *testcase*. Keluaran dari *testcase* dibangkitkan dengan menggunakan solusi juri. Untuk membangkitkan keluaran dari *testcase*, solusi juri dikompilasi kemudian dijalankan dengan memberikan input yang berasal dari *testcase generator*. Keluaran dari solusi juri ini akan digunakan sebagai keluaran dari *testcase*.

Testcase yang sudah terbentuk disimpan pada suatu direktori tertentu. Direktori yang menjadi tempat penyimpanan *testcase* tidak akan dihapus ketika penilaian sudah selesai dilakukan. Hal ini bertujuan agar proses penilaian *job* selanjutnya tidak perlu melakukan pembangkitan *testcase* lagi.

IV.3.2 Penghitungan Waktu Dan *Memory* Solusi Juri

Waktu dan *memory* dari eksekusi solusi juri perlu dihitung untuk menentukan batasan waktu dan *memory* solusi peserta. Waktu dan *memory* eksekusi solusi juri dihitung pada saat pembangkitan *testcase*. Solusi juri diperlukan dalam membangkitkan keluaran dari *testcase*. Pada saat keluaran *testcase* dibangkitkan, lamanya waktu eksekusi dan penggunaan *memory*-nya dihitung dan dicatat.

Perhitungan waktu eksekusi dilakukan dengan menggunakan *UGSbox*. Proses yang dijalankan menggunakan *UGSbox* dapat dihitung penggunaan waktu dan *memory*-nya. Jika solusi juri melebihi batas waktu yang ditentukan oleh juri, maka *UGJob* akan memberikan *verdict internal error* yang menandakan bahwa penilaian gagal dilakukan. Hal ini dapat terjadi ketika komputer peserta sangat lambat.

IV.3.3 Eksekusi Solusi Peserta

Setelah *testcase* berhasil dibangkitkan dan batasan untuk peserta sudah berhasil dihitung, solusi peserta akan mulai dijalankan. Seperti program lainnya, solusi peserta dijalankan menggunakan *UGSbox* untuk menghindari penggunaan *resource* yang berlebihan dan menghindari risiko dari adanya serangan yang ada pada program solusi peserta.

Sebelum solusi peserta dijalankan, tentunya perlu ada proses kompilasi. Proses kompilasi ini dilakukan dengan cara yang sama untuk mengkompilasi program lain. Pada saat kompilasi, beberapa kegagalan mungkin terjadi seperti: kesalahan *syntax*, penggunaan *memory* terlalu besar, dan proses kompilasi berjalan terlalu lama. Jika terjadi kegagalan pada tahap kompilasi, maka *UGJob* akan memberikan *verdict compile error*.

Penggunaan CPU dan *memory* dari program solusi peserta dibatasi berdasarkan batasan yang diperoleh ketika menjalankan program solusi juri. Jika solusi peserta menggunakan *memory* melebihi batasan yang sudah dihitung, maka *UGJob* akan memberikan *verdict memory limit exceeded*. Selain itu, jika solusi peserta berjalan terlalu lama dan melebihi batasan waktu yang sudah dihitung maka *UGJob* akan memberikan *verdict time limit exceeded*. Program solusi peserta mungkin saja melakukan kesalahan yang menyebabkan munculnya *error* seperti pembagian dengan nol, atau mengakses alamat *memory* yang belum dialokasikan. Jika hal ini terjadi, maka *UGJob* akan memberikan *verdict runtime error*.

Keluaran dari program solusi peserta disimpan pada direktori sementara yang diletakkan pada */tmp*. Selanjutnya keluaran dari program solusi peserta akan dinilai dengan cara dibandingkan dengan keluaran solusi juri. Setelah penilaian selesai dilakukan, keluaran program solusi peserta sudah tidak digunakan lagi. Oleh sebab itu, keluaran program solusi peserta dihapus setelah penilaian selesai dilakukan.

IV.3.4 Penilaian Keluaran Solusi Peserta

Keluaran dari program solusi peserta perlu dinilai kebenarannya. Penilaian kebenaran dari program solusi peserta dilakukan dengan cara membandingkannya dengan keluaran solusi juri. *UGJob* membandingkan keluaran program solusi peserta dengan program solusi juri dengan menggunakan *checker*.

Checker merupakan sebuah program yang dibuat oleh juri dan digunakan untuk menilai kebenaran program solusi peserta. Seperti pada program lainnya, perlu adanya proses kompilasi yang dilakukan pada *checker*. Proses kompilasi *checker* dilakukan dengan cara yang sama seperti kompilasi program lainnya.

Kode. IV.3: Contoh Program *Checker*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char** argv) {
    if (argc < 3)
        return -1;

    char* output = argv[1];
    int output_fd = open(output, O_RDONLY);
    if (output_fd < 0) {
        perror("Cannot read output file\n");
        return -1;
    }

    char* expected = argv[2];
    int expected_fd = open(expected, O_RDONLY);
    if (expected_fd < 0) {
        perror("Cannot read expected output file\n");
    }

    const unsigned int buffer_size = 1024 * 4; // using 4KB
    buffer
    char* buffer1 = (char*) malloc(buffer_size);
    char* buffer2 = (char*) malloc(buffer_size);
    int read1, read2;
    do {
        read1 = read(output_fd, buffer1, buffer_size);
```

```

    if (read1 < 0) {
        perror("Cannot read from output file");
        return -1;
    }

    read2 = read(expected_fd, buffer2, buffer_size);
    if (read2 < 0) {
        perror("Cannot read from expected file");
        return -1;
    }

    if (read1 != read2) {
        printf("WA\n"); // wrong answer, file has different
                        size
        return 0;
    }

    if (memcmp(buffer1, buffer2, read1) != 0) {
        printf("WA\n"); // wrong answer, file differ
        return 0;
    }
} while (read1 > 0);

printf("AC\n");
return 0;
}

```

Setelah *checker* selesai dikompilasi, *checker* akan dijalankan dengan memberikan masukan berupa masukan *testcase*, keluaran program solusi juri dan keluaran program solusi peserta. *Checker* kemudian akan memberikan keluaran berupa kebenaran dari program solusi peserta. Kode IV.3 merupakan contoh *checker* yang membandingkan kesamaan tiap *byte* dari keluaran solusi peserta dan solusi juri.

Program *checker* akan memberikan keluaran berupa *string* yang dapat bernilai AC atau WA. Keluaran *checker* akan bernilai AC jika solusi peserta dianggap benar, dan akan bernilai WA jika sebaliknya. *UGJob* kemudian akan mengirimkan *verdict* kepada *UGServer* sesuai dengan hasil penilaian *checker*.

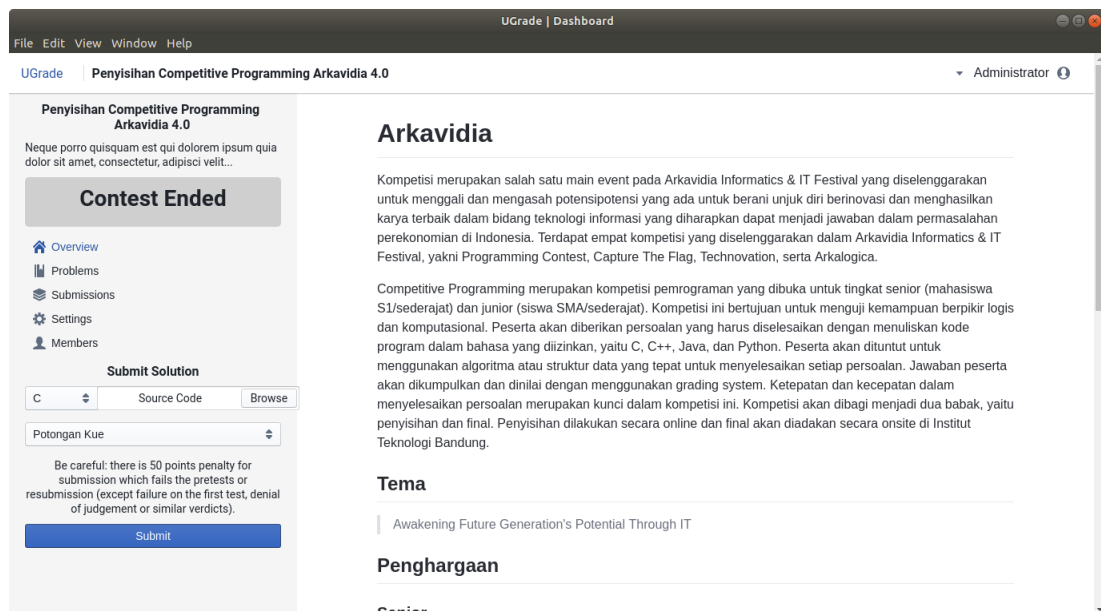
Program *checker* yang dibuat oleh juri mungkin saja memiliki kesalahan. Kesalahan tersebut mengakibatkan program *checker* menghasilkan *error*. Selain itu, jika program *checker* juga mungkin menggunakan terlalu banyak *memory* dan menghabiskan terlalu banyak *waktu*. Jika hal ini terjadi, maka program *checker* akan dihentikan dan *UGJob* akan menghasilkan *verdict internal error*.

Setelah penilaian selesai dilakukan dan *verdict* sudah ditentukan, hasil penilaian akan

dikirmkan ke *UGServer*. Hasil penilaian yang dikirimkan berupa *verdict* hasil penilaian. Pengiriman dilakukan melalui *POST HTTP request*. *Job token* disertakan pada *header* dari *HTTP request* sebagai identitas dari *job* yang telah diselesaikan.

IV.4 *UGDesktop*

Dalam kelompok perangkat lunak *UGrade*, pengguna dapat berinteraksi dengan sistem kompetisi dengan *UGDesktop*. *UGDesktop* memudahkan interaksi pengguna dengan memberikan antar-muka berbasis grafis. Pengguna dapat meng-*install UGDesktop* pada komputernya. Pengguna kemudian dapat menjalankan *UGDesktop* sebagai aplikasi *desktop* pada komputernya.



Gambar IV.4.1: Halaman Kompetisi Dari *UGDesktop*

UGDesktop dikembangkan menggunakan *typescript*, *react* dan *electron*. *React* dipilih karena mudah untuk digunakan dan banyak *library* yang mendukung pengembangan aplikasi berbasis *react*. *React* umumnya digunakan untuk mengembangkan aplikasi berbasis *web* dan membutuhkan *web browser* untuk menjalankan aplikasi tersebut. Pengguna membutuhkan aplikasi yang dapat berjalan sebagai *desktop app* untuk dapat menjalankan sistem antar muka pengguna sekaligus *worker*. Untuk mengatasi hal tersebut, *electron* digunakan untuk menjalankan aplikasi *web* yang berbasis *react* sebagai aplikasi *desktop*. Gambar IV.4.1 merupakan contoh salah satu halaman pada *UGDesktop*.

Untuk melakukan penilaian, *UGDesktop* secara periodik akan meminta menjalankan

UGJob. Dalam menjalankan *UGJob*, *UGDesktop* menggunakan *token* yang didapatkannya ketika pengguna melakukan *sign in*. *UGJob* yang dijalankan *UGDesktop* akan meminta *job* dari *UGServer* untuk dinilai.

IV.5 Pengujian

Pengujian dilakukan dengan menyimulasikan proses pengiriman jawaban oleh pengguna. Sebuah kompetisi diciptakan dengan beberapa pengguna yang akan mengirimkan jawaban secara periodik. Kinerja sistem yang dibangun dibandingkan dengan sistem *online judge* yang bersifat *open source* yaitu *DOMJudge*. *DOMJudge* dipilih karena bersifat *open source* dan sudah populer digunakan untuk menyelenggarakan kompetisi *competitive programming*. *DOMJudge* telah digunakan untuk menyelenggarakan ACM-ICPC, Arkavidia, Vocompfest dan banyak kompetisi lainnya.

Selain pengujian kinerja, diperlukan pengujian kebenaran dari sistem yang dibangun. Pengujian kebenaran dilakukan dengan menyimulasikan proses pengiriman jawaban oleh peserta. Kebenaran sistem yang dibangun ditentukan berdasarkan *verdict* penilaian yang diberikan oleh sistem. Sistem yang benar akan memberikan *verdict* yang sama ketika jawaban yang sama dinilai berkali-kali di komputer yang berbeda. Pada pengujian kebenaran, digunakan sebuah soal dengan enam buah jawaban yang diharapkan akan memiliki *verdict accepted*, *wrong answer*, *memory limit exceeded*, *time limit exceeded*, *runtime error*, dan *compile error*.

Sistem *online judge* yang dibangun perlu memiliki tingkat keamanan yang cukup sehingga tidak membahayakan komputer peserta dan sistem. Pengujian keamanan dilakukan dengan menyimulasikan pengiriman jawaban yang berbahaya ke sistem *online judge*. Jawaban yang dikirimkan adalah sebagai berikut:

1. jawaban yang berisi *fork bomb*
2. jawaban yang berisi *compile bomb*
3. jawaban yang berisi *reverse shell*
4. jawaban yang mencoba keluar dari lingkungan *sandbox*

Komputer yang bertindak sebagai worker diharapkan dapat tetap berjalan setelah jawaban tersebut dieksekusi.

Kerahasiaan dan integritas dari *testcase*, solusi juri dan solusi peserta perlu dijaga. Untuk menjaga kerahasiaan dan integritas, diperlukan adanya pengujian. Pengujian integritas dilakukan dengan melakukan percobaan merusak *testcase*, solusi juri dan

solusi peserta yang ada pada komputer *worker*. Perusakan pada *testcase*, solusi juri ataupun solusi peserta diharapkan dapat dikenali dan diatasi sehingga hasil penilaian dari *worker* akan tetap benar. Kerahasiaan *testcase*, solusi juri dan solusi peserta hanya akan diuji dengan melihat isi dari *file testcase*, solusi juri dan solusi peserta. Kerahasiaan akan dianggap lolos uji jika isi dari *file testcase*, solusi juri maupun solusi peserta tidak dapat dibaca oleh pengguna.

BAB V

SIMPULAN DAN SARAN

Pada bab ini diberikan kesimpulan dari Tugas Akhir dan saran pengembangan lebih lanjut terhadap solusi yang sudah dipaparkan.

V.1 Simpulan

Pada tugas akhir ini telah diciptakan sistem *online judge* yang bernama *UGrade*. *UGrade* memanfaatkan komputer pengguna untuk menjadi *worker* dalam melakukan penilaian jawaban peserta kompetisi. Peningkatan kinerja dari sistem yang dibangun belum dapat ditentukan karena pengujian belum dilakukan.

Pengujian terhadap serangan yang mungkin dilakukan oleh peserta dilakukan dengan mengirimkan jawaban yang berbahaya kepada sistem yang dibangun. Keamanan dari sistem belum dapat ditentukan karena pengujian belum ditentukan.

Kebenaran dari sistem penilaian diuji dengan mengirimkan jawaban secara berulang-ulang dengan harapan *autograder* memberikan *verdict* yang sama. Kebenaran dari sistem yang dibangun belum dapat ditentukan karena pengujian belum dilakukan.

V.2 Saran

Saran yang dapat diberikan untuk pengembangan terkait pengerjaan tugas akhir ini adalah sebagai berikut:

1. Pemilihan teknik penilaian membutuhkan adanya studi lebih lanjut. Jika banyak peserta melakukan serangan terhadap sistem, *self grading* dapat mengurangi risiko terjadinya kerusakan sistem. Sistem *online judge* dapat dibuat mudah untuk dikonfigurasi sehingga dapat menggunakan beberapa jenis sistem penilaian.

2. *UGDesktop* berukuran sangat besar karena mengguna *framework* React dan Electron. Aplikasi *desktop* dapat dibuat lebih *native* dengan menggunakan *library* seperti GTK.
3. Pemilihan *framework* Django untuk bertindak sebagai *server* tidak dapat menangani banyaknya pengguna dengan efisien. Bahasa Go dapat digunakan untuk meningkatkan kinerja dari *server*.
4. Penggunaan bahasa Go untuk mengembangkan lingkungan *sandbox* kurang efektif. Bahasa Go sulit untuk melakukan *system call fork* dan *exec* secara terpisah karena adanya *go runtime* yang perlu dijalankan. *Sandbox* dapat dikembangkan dengan bahasa yang lebih *low-level* seperti C, C++ atau Rust.
5. Penggunaan istilah *grading group*, *grading size*, *grading* dan *grader group* sulit dipahami. Penggunaan nama yang mirip sebaiknya dihindari dengan memilih nama yang lebih deskriptif dan tidak terlalu mirip.
6. Bahasa yang didukung oleh *UGrade* masih terbatas C dan C++. *UGrade* perlu memberikan dukungan terhadap bahasa Pascal, Python dan Java karena bahasa tersebut sering digunakan pada kompetisi *competitive programming*.

DAFTAR PUSTAKA

- Bez, J. L., N. A. Tonin, & P. R. Rodegheri (Agustus 2014). “URI Online Judge Academic: A tool for algorithms and programming classes”. In: *2014 9th International Conference on Computer Science Education*, pp. 149–152. DOI: 10.1109/ICCSE.2014.6926445.
- Danutama, Karol & Inggriani Liem (2013). “Scalable Autograder and LMS Integration”. In: *Procedia Technology* 11. 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013, pp. 388–395. ISSN: 2212-0173. DOI: <https://doi.org/10.1016/j.protcy.2013.12.207>.
- Felter, W. et al. (Mar. 2015). “An updated performance comparison of virtual machines and Linux containers”. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172. DOI: 10.1109/ISPASS.2015.7095802.
- Fernando, Jordan & Inggriani Liem (2014). “Components and Architectural Design of an Autograder System Family”. In:
- Halim, S. & F. Halim (2013). *Competitive Programming*.
- ICPC Foundation (2018a). *ICPC Fact Sheet*. URL: <https://icpc.baylor.edu/worldfinals/pdf/Factsheet.pdf>.
- (2018b). *World Finals Rules for 2019*. URL: <https://icpc.baylor.edu/worldfinals/rules>.
- IOI Organization (2017a). *IOI 2017 Contest Rules*. URL: <http://ioi2017.org/contest/rules/>.
- (2017b). *Organization*. URL: <https://ioinformatics.org/page/organization/4>.
- Merkel, Dirk (Mar. 2014). “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux J*. 2014.239. ISSN: 1075-3583. URL: <http://dl.acm.org/citation.cfm?id=2600239.2600241>.
- Mirzayanov, M. (2011). *Codeforces Contest Rules*. URL: <http://codeforces.com/blog/entry/4088>.
- P., Lessard (2003). “InfoSec reading room report: Linux process containment - a practical Look at chroot and user mode Linux”. In:

- Stevenson, J. (Sept. 2018). *MDN WebAssembly*. URL: <https://developer.mozilla.org/en-US/docs/WebAssembly>.
- Wasik, Szymon et al. (2018). “A Survey on Online Judge Systems and Their Applications”. In: *ACM Comput. Surv.* 51.1, 3:1–3:34. ISSN: 0360-0300. DOI: 10.1145/3143560. URL: <http://doi.acm.org/10.1145/3143560>.