API of the multi-tenant SaaS:

The interface in C++ is as follows:

```
#ifndef SaaS_app_H
#define SaaS_app_H
class SaaS_API;
#include "SaaS_API.h"
#include "yaml-cpp/yaml.h"

class Saas_application {
  public:

    Saas_application();

    void print_application_config();

    bool get_multi();
    int get_mem_intensity(int);
    int get_io_intensity(int);
    int get_cpu_intensity(int);
    int get_cache_size();

    void set_mem_intensity(int, int);
    void set_io_intensity(int, int);
    void set_cpu_intensity(int, int);
    void set_cache_size(int);

    int single_tenant_request();
    int multi_tenant_request(int);

  private:
    std::vector<int> cache;
    YAML::Node config;
    void simulate(int, int, int);
```

```
    void tenant_lookup(int);
};

#endif
```

The REST API is defined as follows:

```
#include <string>

#include <iostream>
#include "SaaS_API.h"
#include "lib/crow_all.h"
#include "Memory_stress.h"

SaaS_API::SaaS_API(Saas_application* application):
  application(application)
{}

// /set_mem/id/int
void SaaS_API::setter_api(crow::SimpleApp& app) {

  CROW_ROUTE(app, "/set_mem/<int>/<int>")
    ([this](int id, int mem_intensity) {
     CROW_LOG_INFO << "Setting mem param: " << mem_intensity;
     application->set_mem_intensity(id, mem_intensity);
     return "mem param has been set";
    });

  CROW_ROUTE(app, "/set_io/<int>/<int>")
    ([this](int id, int io_intensity) {
     CROW_LOG_INFO << "Setting io param: " << io_intensity;
     application->set_io_intensity(id, io_intensity);
     return "io param has been set";
    });

  CROW_ROUTE(app, "/set_cpu/<int>/<int>")
    ([this](int id, int cpu_intensity) {
```

```
30      CROW_LOG_INFO << "Setting cpu param: " << cpu_intensity;
31      application->set_cpu_intensity(id, cpu_intensity);
32      return "cpu param has been set";
33     });
34
35   CROW_ROUTE(app, "/set_cache/<int>")
36     ([this](int cache_size) {
37      CROW_LOG_INFO << "Setting cache param: " << cache_size;
38      application->set_cache_size(cache_size);
39      return "cache param has been set";
40     });
41 }
42
43 void SaaS_API::multitenant_api(crow::SimpleApp& app) {
44
45   CROW_ROUTE(app, "/request/<int>")
46     ([this](int tenant_id) {
47      // Maybe do not allow requests for id 0, as it is used as a reserved default id
48      CROW_LOG_INFO << "Multitentant request for id: " << tenant_id;
49      application->multi_tenant_request(tenant_id);
50      return "succes";
51     });
52 }
53
54 void SaaS_API::single_api(crow::SimpleApp& app) {
55   CROW_ROUTE(app, "/request/")
56     ([this]() {
57      CROW_LOG_INFO << "Single tenant request";
58      application->single_tenant_request();
59      return "succes \n";
60     });
61 }
62
63 void SaaS_API::expose() {
64   crow::SimpleApp app;
```

```
65    crow::logger::setLogLevel(crow::LogLevel::CRITICAL);
66    setter_api(app);
67    if (application->get_multi()) {
68      multitenant_api(app);
69    } else {
70      single_api(app);
71    }
72    app.port(5000).multithreaded().run();
73 }
```

All requests are GET requests

To send a message to the API you have to use a GET request, or simply type in your browser, for example the following URL, http://<kubeserviceip>/setmem/0/25. In linux the following command does the job `wget http://<kubeserviceip>/setmem/0/25`

/setmem/0/250 means you run the application as a single tenant and you set the time intensity of the algorithm for the entire application to 25

`wget http://<kubeserviceip>/request` is basically the invocation of a single request

Using pyton, such request can be sent using the package urllib as follows:

```
self.path = "/request/" + str(tenant.tenant_id)
self.request_url = "http://" + self.ip + ":" + self.port + self.path

try:
            urllib.request.urlopen(self.request_url).read()
except:
            self.timeout_count += 1
            print("timeout + 1: " + str(self.timeout_count))
```

The example-controller in github already implements a Request class that uses a Java REST client to invoke the API.

The semantics of the API is best described by reading the thesis. For full reference, here's the implementation of the API

It distinguishes between running the application as a multi-tenant application or as a single tenant application.

```
Saas_application::Saas_application()
14 {
15   config = YAML::LoadFile("saas_config.yaml");
```

```
16
17    SaaS_API* api = new SaaS_API(this);
18    api->expose();
19  }
20
21  int Saas_application::single_tenant_request() {
22    simulate(get_mem_intensity(0), get_cpu_intensity(0), get_io_intensity(0));
23  }
24
25  int Saas_application::multi_tenant_request(int tenant_id) {
26    tenant_lookup(tenant_id);
27    // TODO: getters op basis van id maken.
28    simulate(get_mem_intensity(tenant_id), get_cpu_intensity(tenant_id),
      get_io_intensity(tenant_id));
29  }
30
31  void Saas_application::tenant_lookup(int tenant_id) {
32
33    if (std::find(cache.begin(), cache.end(), tenant_id) == cache.end()) {
34      //perform I/O
35      std::ifstream in("saas_config.yaml");
36      if (in.is_open())
37      {
38        CROW_LOG_INFO << "Id not in cache";
39        std::string line;
40        while ( getline(in,line) )
41        {
42          if (line.find("multi") == 0 ){
43            CROW_LOG_INFO << "Performing I/O";
44          }
45        }
46        in.close();
47      }
48
49    // Add to cache
50      if(cache.size() >= get_cache_size()) {
```

```
      cache.erase(cache.begin());
    }
    cache.push_back(tenant_id);
    std::cout << cache.size() << std::endl;
  }
}

void Saas_application::simulate(int mem, int cpu, int io) {
  StressMemory* mem_stress = new StressMemory(mem);
  Cpu_stress* cpu_stress = new Cpu_stress(cpu);
  Io_stress* io_stress = new Io_stress(io);

  mem_stress->run();
  cpu_stress->run();
  io_stress->run();
  mem_stress->release();
  delete mem_stress;
  delete cpu_stress;
  delete io_stress;
}

void Saas_application::print_application_config() {

  std::cout << "multitenancy: "<<  get_multi() << std::endl;
  std::cout << "mem_intensity: " <<  get_mem_intensity(0) << std::endl;
  std::cout << "cpu_intensity: " <<  get_cpu_intensity(0) << std::endl;
  std::cout << "io_intensity: "  <<  get_io_intensity(0) << std::endl;

}

bool Saas_application::get_multi() {
  return config["multi"].as<bool>();
}

int Saas_application::get_mem_intensity(int id) {
  if (config["tenants"][id]) {
    return config["tenants"][id]["mem_intensity"].as<int>();
```

```
  } else {
    return config["mem_intensity"].as<int>();
  }
}

int Saas_application::get_io_intensity(int id) {
  if (config["tenants"][id]) {
    return config["tenants"][id]["io_intensity"].as<int>();
  } else {
    return config["io_intensity"].as<int>();
  }
}

int Saas_application::get_cpu_intensity(int id) {
  if (config["tenants"][id]) {
    return config["tenants"][id]["cpu_intensity"].as<int>();
  } else {
    return config["cpu_intensity"].as<int>();
  }
}

int Saas_application::get_cache_size() {
  return config["cache_size"].as<int>();
}


void Saas_application::set_mem_intensity(int id, int mem_int) {
  CROW_LOG_INFO << "setting mem_intensity " << mem_int << " for id: " << id;
  if (config["tenants"][id]) {
    CROW_LOG_DEBUG << "setting for id: " << id;
    config["tenants"][id]["mem_intensity"] = mem_int;
  } else if (id == 0) {
    CROW_LOG_DEBUG << "setting default";
    config["mem_intensity"] = mem_int;
  } else {
```

```
123       CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
124     }
125 }
126
127 void Saas_application::set_io_intensity(int id, int io_int) {
128     CROW_LOG_INFO << "setting io_intensity " << io_int << " for id: " << id;
129     if (config["tenants"][id]) {
130       config["tenants"][id]["io_intensity"] = io_int;
131     } else if (id == 0) {
132       config["io_intensity"] = io_int;
133     } else {
134       CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
135     }
136 }
137
138 void Saas_application::set_cpu_intensity(int id, int cpu_int) {
139     CROW_LOG_INFO << "setting cpu_intensity " << cpu_int << " for id: " << id;
140     if (config["tenants"][id]) {
141       config["tenants"][id]["cpu_intensity"] = cpu_int;
142     } else if (id == 0) {
143       config["cpu_intensity"] = cpu_int;
144     } else {
145       CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
146     }
147 }
148
149 void Saas_application::set_cache_size(int cache_size) {
150     config["cache_size"] = cache_size;
151 }
```