# IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang implementasi sistem sesuai dengan analisa dan desain sistem. Implementasi sistem meliputi installasi teknologi yang diperlukan, pengaturan *database* MySQL dan implementasi aplikasi.

## Installasi dan Konfigurasi Gitlab

Langkah pertama untuk implementasi system adalah installasi dan konfigurasi *Gitlab* pada server yang ada. Server yang digunakan adalah Ubuntu 16.06 LTS, RAM 8 GB, dan Hardisk 500GB

### Installasi Gitlab

Tahapan dalam installasi *Gitlab* adalah:

1. Update dan installasi komponen yang diperlukan

**sudo** apt-get update

**sudo** apt-get install ca-certificates curl openssh-server  
**sudo** apt-get install postfix

1. Download bash installer

**cd** **/**tmp

**curl** **-**LO https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh

1. Setup

**sudo** bash **/**tmp**/**script.deb.sh

1. Install

**sudo** apt-get install gitlab-ce

1. Konfigurasi Ulang

**sudo** gitlab-ctl reconfigure

### Konfigurasi Gitlab

Kofigurasi *Gitlab* yang dilakukan adalah mengubah *listen port.* Pada umumnya *gitlab* menggunakan / *lister* *port* 80 sebagai *port default* setelah melakukan installasi. *Port* 80 ini akan digunakan sebagai *port* untuk aplikasi web yang akan digunakan, maka *port* *gitlab* akan dirubah menjadi *port* 8888.

Cara melakukan konfigurasi:

1. Buka file konfigurasi *gitlab*

**sudo** vi **/**etc**/**gitlab**/**gitlab.rb

1. Ubah data port menjadi 8888

external\_url 192.168.37.30:8888

1. Konfigurasi Ulang

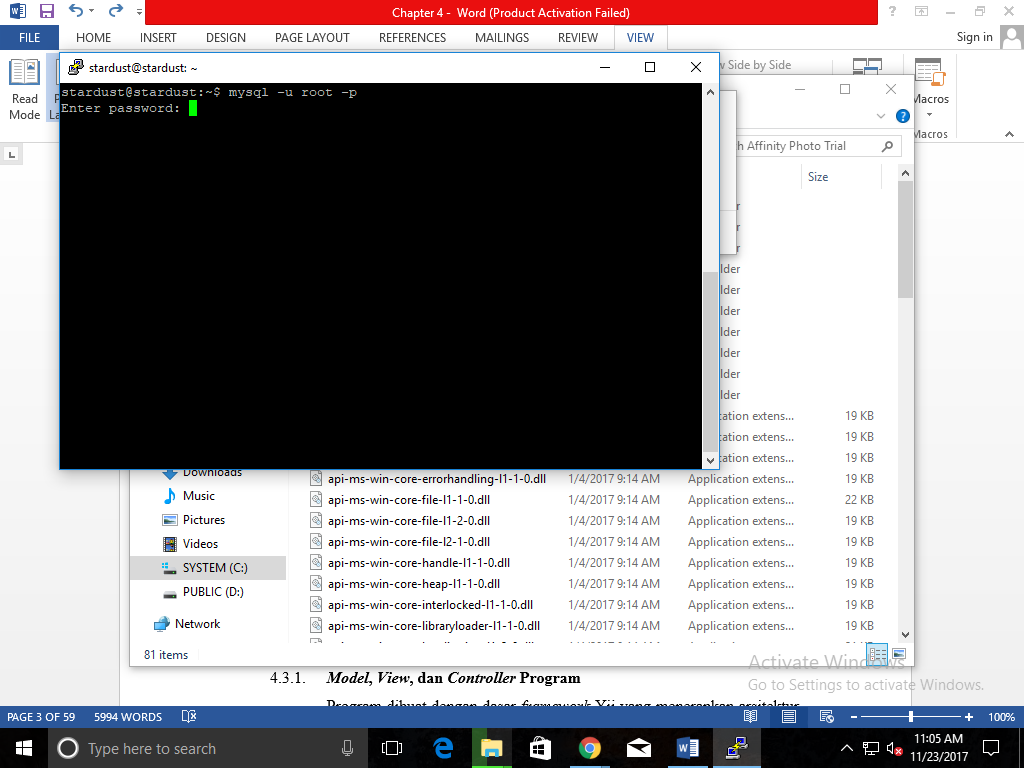
**sudo** gitlab-ctl reconfigure

Setelah melakukan konfigurasi ulang, akses *gitlab* menggunakan *web browser* dengan menuju alamat http://192.168.37.30:8888. Pada halaman awal *website*, *website* akan meminta untuk mengisikan *password* untuk *root user*. Setelah password untuk *root user* terisi maka website *gitlab* dapat diakses dan dapat membuat *user* baru (*register*).

## Pengaturan *Database*

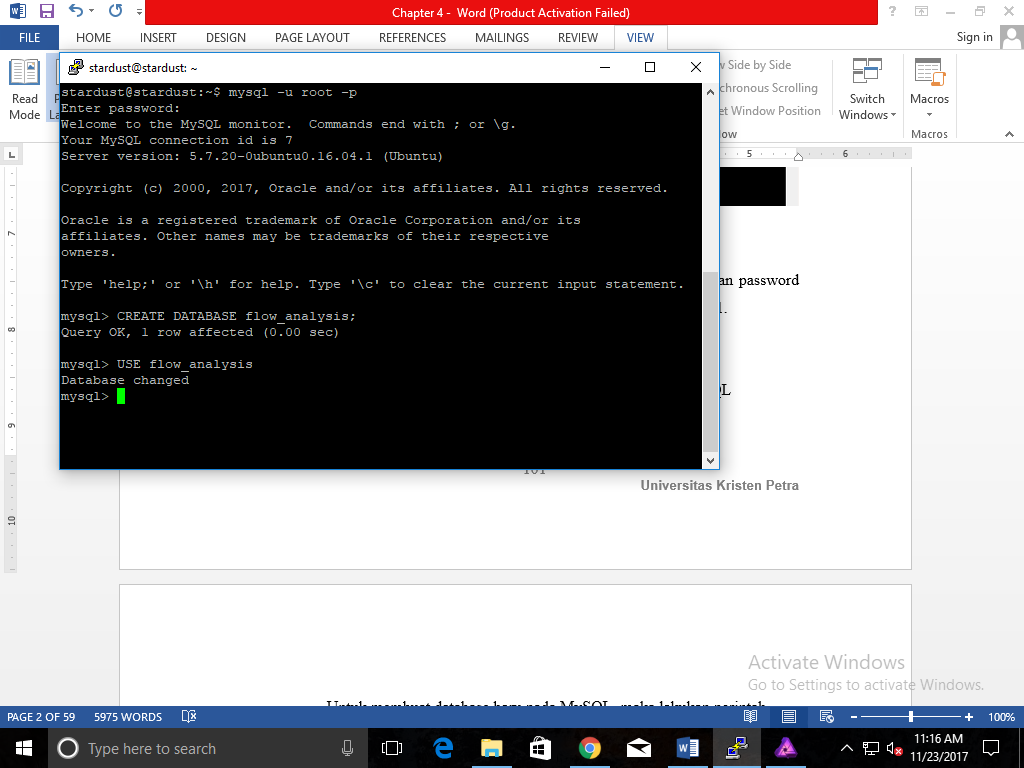
### Membuat *Database* Baru

Langkah pertama dalam melakukan implementasi sistem adalah membuat database baru untuk menyimpan data – data yang diperlukan sistem. Database yang digunakan adalah MySQL, pertama lakukan koneksi menuju database MySQL pada server.



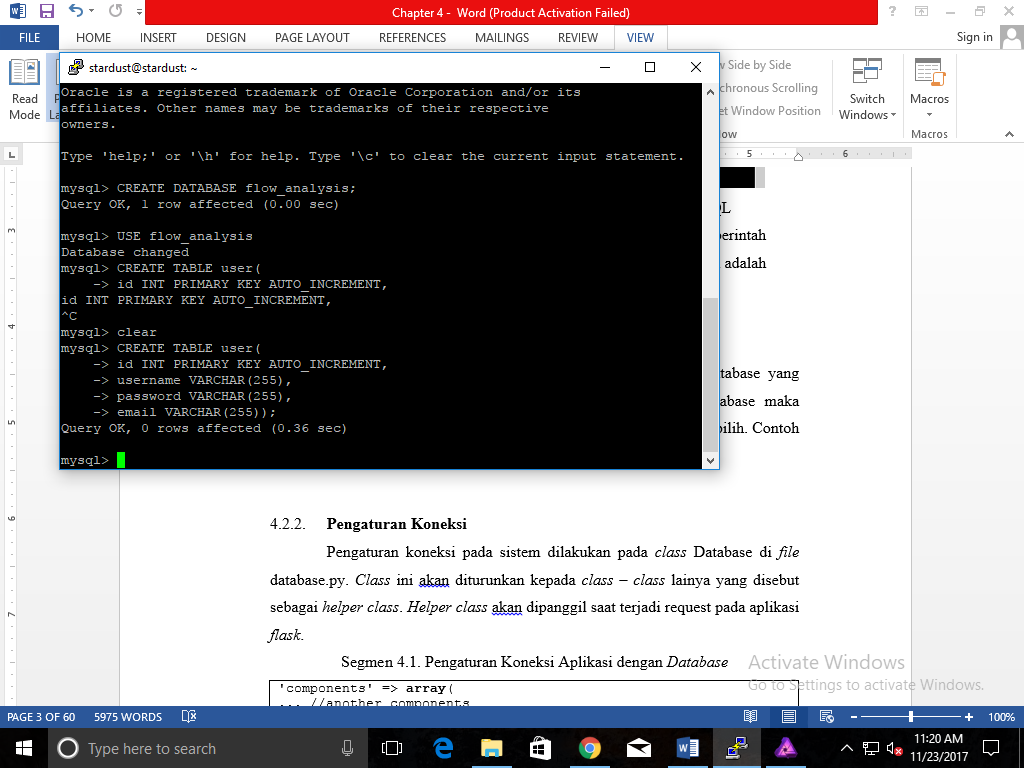
Tampilan Koneksi *Server*

Untuk masuk kedalam database maka masukan username dan password dengan menggunakan *command* atau perintah seperti pada Gambar 4.1.



Perintah Membuat *Database* Baru pada MySQL

Untuk membuat database baru pada MySQL, maka lakukan perintah pada Gambar 4.2.2. Nama database yang digunakan pada tugas akhir ini adalah flow\_analysis.



Tampilan Eksekusi SQL pada MySQL

Untuk melakukan query pada MySQL, permata pilih database yang tersedia dengan menggunakan perintah *use*. Setelah memilih database maka perintah SQL dapat dijalankan untuk memanipulasi database yang terpilih. Contoh menjalankan SQL pada MySQL terdapat pada Gambar 4.3.

### Pengaturan Koneksi

Pengaturan koneksi pada sistem dilakukan pada *class* Database di *file* database.py. *Class* ini akan diturunkan kepada *class* – *class* lainya yang disebut sebagai *helper* *class*. *Helper* *class* akan dipanggil saat terjadi request pada aplikasi *flask*.

Pengaturan Koneksi Aplikasi dengan *Database*

|  |
| --- |
| **import** pymysql  **class** Database:con = None  cur = None   **def** \_\_init\_\_(self):self.con = pymysql.connect(host=**'localhost'**,  user=**'root'**,  password=**''**,  db=**'flow\_analysis'**)  self.cur = self.con.cursor(pymysql.cursors.DictCursor) |

## Implementasi Program

Implementasi program terdiri dari beberapa tahap, yaitu pembuatan helper class yang merupakan turunan / *inheritance* dari class database. Kemudian implementasi dilanjutkan dengan pembuatan *view* dengan menggunakan *jinja* *template*, dan terakhir menggabungkan *view* dan *helper* pada aplikasi utama (main.py).

### *Flask, Helper Class* dan *Database Class*

Program dibuat menggunakan flask microframework. *Flask* merupakan *microframework* dimana cara pekodean untuk membuat sebuah tampilan website sangat singkat dan mudah. Karena karakteristiknya, *flask* dapat dikembangkan sesuai keinginan. Karena pekodean yang singkat maka diperlukan *class* – *class* lainya yang bertugas sebagai *controller* dan *model* yang mengatur urutan kerja / logika proses dan pengolahan data. Class – class yang membantu implementasi ini disebut sebagai *helper* *class* yang merupakan turunan dari *database* *class*. *Database* *class* bertugas sebagai penyedia data sedangkan *helper* *class* bertugas untuk memberikan urutan kerja atau logika proses yang harus dijalankan.

Daftar *File* *Helper Class* dalam Program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *File* | *Inherit From* | Keterangan | ERD | Activity |
| user.py | Database  (database.py) | Helper class untuk pengaturan user / pengguna | user | Login |
| Add Enterprise Administrator |
| Add Enterprise User |
| Edit Enterprise Administrator |
| Edit Enterprise User |
| Delete Enterprise Administrator |
| Delete Enterprise User |
| enterprise.py | Database  (database.py) | Helper class untuk pengaturan data enterprise | enterprise | Add Enterprise |
| Edit Enterprise |
| Delete Enterprise |
| File.py | Database  (database.py) | Helper class untuk pengaturan data enterprise | files | Upload File |
| Download File |
| shared\_file | Share File |
| Unshare File |
| recycle | Delete File |
| Purge File |
| Recover File |

Daftar *File* *Helper Class* dalam Program (lanjutan)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *File* | *Inherit From* | Keterangan | ERD | Activity |
| project.py | Database  (database.py) | Helper class untuk pengaturan data project | project | Add Project |
| Manage Project |
| Save Project |
| Load Project |

### Permbuatan List API

Pada setiap tampilan / *view* aplikasi akan melakukan akses menuju API (*Aplication Program Interface*) yang telah disediakan. Hal ini bertujuan untuk mempermudah pengembangan aplikasi. Pada setiap *view* terdapat potongan program *javascript*/*jQuery* yang melakukan *request* pada API yang tersedia untuk melakukan manipulasi data. API yang digunakan berdasarkan REST *(Representative of State*) dengan mengembalikan data (*return*) berupa JSON (*Javascript Object Notation*).

Tabel API Berdasarkan *Helper Class* (*user & enterprise*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *File* | URL | *Method* | *Payload* | Keterangan |
| user.py | /api/users/<id> | GET | None | Mengambil data seorang *user* dengan id yang ditentukan |
| /api/users | POST | * Username * Password * Confirm * Email | Menambahkan seorang *user* kedalam system sesuai dengan data yang ada dalam *payload* |
| /api/users | DELETE | None | Menghapus *user* sesuai dengan id yang ditentukan |
| user.py & enterprise.py | /api/users/<id> | PUT | * Password * Confirm * Enterprise Name * Email * Phone * Address * Filesize * User\_limit | Mengubah data seorang *user* yang dikenali dengan id yang sudah di tentukan. Data berubah sesuai dengan *payload* yang ada. |

Tabel API Berdasarkan *Helper Class* (*file*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *File* | URL | *Method* | *Payload* | Keterangan |
| file.py | /api/upload | POST | * *File* | Menerima file dari client dalam berntuk HTTP POST dan meletakan file pada *home\_directory* user berdasarkan *session* yang sedang aktif |
| /api/download/<id\_file> | GET | *None* | Mengirim file kepada *client.* Pada *client* file diterima dalam bentuk *download* |
| /api/recycle | POST | * *File ID* | Mengirimkan *file* pada *Enterprise Recycle Bin* |
| /api/shareable/<id\_file> | GET | *None* | Mencari *user* yang dapat di-*share* berdasarkan *file* yang ingin di-*share* |
| /api/share | POST | * *File ID* * *User ID* * *Permission* | Memberikan hak akses *file* kepada *user* berdasarkan *User ID* pada *payload.* |

Tabel API Berdasarkan *Helper Class* (*file*) Lanjutan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *File* | URL | *Method* | *Payload* | Keterangan |
| file.py | /api/share/detail/<id\_file> | GET | *None* | Memberika data *detail file* berdasarkan *id\_file* pada *parameter*. Data meliputi siapa saja yang memiliki hak akses pada *file* tersebut. |
|  | /api/share | DELETE | *None* | Menghapus hak akses pengguna untuk sebuah file (*unshare*) |
|  | /api/file/<id\_file> | DELETE | *None* | Menghapus *file* dari *system* dan dari *database* berdasarkan *id* yang diberikan. |
|  | /api/recycle/<id\_recycle> | DELETE | *None* | Menghapus *file* pada *list file* di *Enterprise* *Recycle Bin*. *File* akan kembali ke pemilik file. (*Restore File*) |

### Pembuatan *Services*

#### Konfigurasi

Konfigurasi dilakukan untuk memberikan parameter – parameter yang dibutuhkan untuk beberapa fitur yang ada pada *flask*. Konfigurasi yang di perlukan adalah lokasi *folder* *upload*, ekstensi *file* yang diperbolehkan untuk di *upload, secret key* (digunakan untuk *session*), dan yang terakhir adalah pengaturan *cross origin* header yang diperbolehkan jika melakukan akses terhadap URL dengan pola dimulai dengan ‘/api/’

Konfigurasi Flask

|  |
| --- |
| UPLOAD\_FOLDER = **'/upload'** ALLOWED\_EXTENSIONS = set([**'csv'**])  app = Flask(\_\_name\_\_)  app.config[**'UPLOAD\_FOLDER'**] = UPLOAD\_FOLDER app.secret\_key = **"bluebutterfly"** cors = CORS(app, resources={**r"/api/\*"**: {**"origins"**: **"\*"**}}) |

#### *User Helper Services*

*User helper services* merupakan kumpulan REST API yang memanfaatkan *User helper class* dan *Enterprise helper class*. Layanan – layanan yang tersedia dipanggil pada *view* dengan menggunakan *Javascript / jQuery*.

Fungsi *get\_user*

|  |
| --- |
| @app.route(**"/api/users/<id>"**, methods=[**'GET'**]) **def** get\_user(id):  user\_helper = user.User()   **if** session.get(**"login"**) **is** None:  abort(403)   id\_session = session[**'id'**]  user\_helper.load\_user(id\_session)  data = user\_helper.get\_data()   **if** data[**'auth'**] == **'user'**:  abort(403)  **if** id **is not** None:  res = user\_helper.load\_user(id)  **if** res **is** False:  abort(404)  **if** res[**'auth'**] == **'root'**:  abort(403)  **if** res[**'auth'**] == **'admin'**:  enterprise\_helper = enterprise.Enterprise()  res[**'enterprise'**] = enterprise\_helper  .get\_enterprise\_by\_user(res[**'id'**])  **return** jsonify(res)  **else**:  **return** jsonify({**"err"**: 1, **"msg"**: **"Must Have ID"**}) |

Fungsi pada Segmen 4.3 bertugas untuk mencari pengguna atas id yang diberikan sebagai parameter.

Fungsi *add\_user*

|  |
| --- |
| @app.route(**"/api/users"**, methods=[**'POST'**]) **def** add\_user():  user\_helper = user.User()   err = 0  msg = **''   if** request.method == **'POST'**:  username = request.form[**'username'**]  email = request.form[**'email'**]  password = request.form[**'password'**]  confirm = request.form[**'confirm'**]   auth = **''** admin = 0  **if** session[**'auth'**] == **'root'**:  auth = **'admin'  elif** session[**'auth'**] == **'admin'**:  auth = **'user'** admin = session[**'id'**]  **if** confirm **is not** password:  err = 1  msg = **"Confirm Password Incorrect"   if** user\_helper  .add\_user(username, password, email, auth, admin):  **if** auth == **"admin"**:  *# Add Enterprise Data* name = request.form[**'enterprise\_name'**]  address = request.form[**'enterprise\_address'**]  email = request.form[**'enterprise\_email'**]  phone = request.form[**'enterprise\_phone'**]  filesize = request.form[**'enterprise\_filesize'**]  user\_limit = request.form[**'enterprise\_user'**]  enterprise\_helper = enterprise.Enterprise()   u = user\_helper.get\_by\_username(username)   **if** enterprise\_helper  .add\_enterprise(name,  address,  email,  phone,  filesize,  user\_limit,  u[**'id'**]):  err = 0  msg = **"Success"  else**:  err = 1  msg = **"Failed"  else**:  err = 1  msg = user\_helper.err\_msg()  **return** jsonify({  **'err'**: err,  **'msg'**: msg  }) |

Fungsi pada Segmen 4.4 digunakan untuk menambahkan user kedalam sistem. HTTP Method yang digunakan adalah POST.

Fungsi *delete\_user*

|  |
| --- |
| @app.route(**"/api/users/<id>"**, methods=[**'DELETE'**]) **def** delete\_user(id):  user\_helper = user.User()   **if** session.get(**"login"**) **is** None:  abort(403)   id\_session = session[**'id'**]  user\_helper.load\_user(id\_session)  data = user\_helper.get\_data()   **if** data[**'auth'**] == **'user'**:  abort(403)   user\_helper.load\_user(id)  res = user\_helper.get\_data()  **if** res[**'auth'**] == **'user'**:  f = fl.Files()  f.transfer\_protocol(id, res[**'admin'**])  **if** user\_helper.delete\_user(id):  **return** jsonify({**"err"**: 0, **"msg"**: **"Success"**}) **else**:  **return** jsonify({**"err"**: 1, **"msg"**: user\_helper.err\_msg()}) |

Fungsi pada Segmen 4.5 bertujuan untuk menghapus *user* sberseta data – data yang bersangkutan dengan *user* tersebut. Semua file yang dimiliki *user* akan diberikan kepada Enterprise Administrator perusahaan *user* tersebut*.*

Fungsi *update\_user*

|  |
| --- |
| @app.route(**"/api/users/<id>"**, methods=[**'PUT'**]) **def** update\_user(id):  user\_helper = user.User()   password = request.form[**'password'**]  confirm = request.form[**'confirm'**]   **if** password != **''**:  **if** password == confirm:  **if not** user\_helper.change\_password(id, password):  **return** jsonify({**"err"**: 1,  **"msg"**: user\_helper.err\_msg()})  **else**:  **return** jsonify({**"err"**: 1,  **"msg"**: **"Detected Password Change but Wrong**  **Confirmation Password"**}) |

Fungsi *update\_user* (lanjutan)

|  |
| --- |
| user\_helper.load\_user(id)  data = user\_helper.get\_data()   **if** data[**'auth'**] == **'admin'**:  name = request.form[**'name'**]  email = request.form[**'email'**]  phone = request.form[**'phone'**]  address = request.form[**'address'**]  filesize = request.form[**'filesize'**]  user\_limit = request.form[**'user\_limit'**]   enterprise\_helper = enterprise.Enterprise()  **if not** enterprise\_helper.update\_enterprise(data[**'id'**],  name,  address,  email,  phone,  filesize,  user\_limit):  **return** jsonify({**"err"**: 1,  **"msg"**: enterprise\_helper.error\_msg()})   **return** jsonify({**"err"**: 0, **"msg"**: **"Success"**}) |

Fungsi Segmen 4.6 digunakan untuk merubah data *user*, jika *user* merupakan *Enterprise Administrator* maka *user* juga dapat mengubah data *Enterprise*.

#### *File Helper Services*

*File helper services* merupakan kumpulan REST API yang memanfaatkan *File helper class*. Layanan – layanan yang tersedia dipanggil pada *view* dengan menggunakan *Javascript / jQuery*.

Fungsi *add\_files*

|  |
| --- |
| @app.route(**"/api/upload"**, methods=[**'POST'**]) **def** add\_files():  enterprise\_helper = enterprise.Enterprise()**if 'file' not in** request.files:  **return** jsonify({  **"err"**: 1,  **"msg"**: **"No File Part "** + str(request.files)  })  file = request.files[**'file'**]**if** file.filename == **''**:  **return** jsonify({  **"err"**: 1,  **"msg"**: **"No Selected File"** }) |

Fungsi *add\_files* (lanjutan)

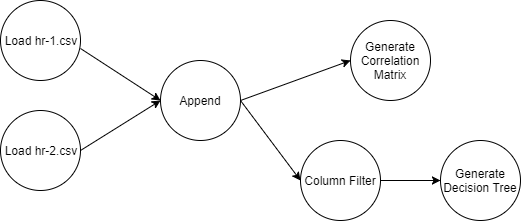
|  |
| --- |
| **if** file **and** allowed\_file(file.filename):  file\_helper = fl.Files()  user\_helper = user.User()  user\_helper.load\_user(session[**'id'**])  user\_data = user\_helper.get\_data()  filename = secure\_filename(file.filename)  basedir = os.getcwd()  **if not** os.path.exists(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**]):  os.makedirs(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**])  **if not** os.path.exists(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**] +  **"/files"**):  os.makedirs(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**] + **"/files"**) **if not** os.path.exists(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**] +  **"/project"**):  os.makedirs(basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**] +  **"/project"**)  target = basedir +  app.config[**'UPLOAD\_FOLDER'**] + **"/"** +  user\_data[**'home\_folder'**] + **"/files"** file.save(os.path.join(target, filename))  file\_helper.add\_file(filename,  session[**'id'**],  user\_data[**'home\_folder'**]  + **"/files/"** + filename)  **return** jsonify({  **"err"**: 0,  **"msg"**: **"Upload File Success"**,  **"filename"**: file.filename  })  **else**:  **return** jsonify({  **"err"**: 1,  **"msg"**: **"Extension not Allowed"** }) |

Fungsi pada Segmen 4.8 bertujuan untuk menambahkan file pada sistem. File dikirim dalam bentuk HTTP POST.

### Implementasi DAG (*Directed Acyclic Graph*)

*Directed Acyclic Graph* merupakan sebuah graph yang memiliki urutan yang linear dan tidak memiliki siklus. Sebuah *node* dari graph tidak akan kembali menuju *node* sebelumnya sehingga tidak terjadi siklus di dalam *graph*. Contoh dari DAG adalah pilihan matakuliah dari semester 1 sampai semester 8 berserta persyaratanya.

Aplikasi ini menggunakan DAG sebagai tampilan *user interface* untuk melakukan pre-processing data sampai processing data. DAG dipilih sebagai metode yang paling baik untuk merepresentasikan alur data karena tidak memiliki siklus di dalam implementasi *graph* dan *graph* memiliki alur yang linear. Alur linear dari DAG dapat memperjelas dan memudahkan pengguna dalam membuat alur data yang bersifat *Input, Process, Output.*



Gambar 4.3.1 Contoh *Directed Acyclic Graph* pada Aplikasi

Ketika aplikasi menjalankan service /api/run, maka aplikasi akan menerima input berupa data dengan format JSON. Data tersebut merupakan metadata yang menunjukan *id* proses, parameter dan alur selanjutnya yang harus dilalui oleh proses.

Pertama aplikasi akan mencari semua proses yang memiliki tipe input kemudian proses akan masuk kedalam sebuah *queue*, pada contoh Gambar 4.3.1 proses yang masuk kedalam *queue* adalah *Load hr-1.csv* dan *Load hr-2.csv*. Kegiatan ini dilakukan untuk menjalankan alur pada graph dengan menggunakan metode BFS (*Breadth-first search*).

|  |  |
| --- | --- |
| **Load hr-1.csv** | **Load hr-2.csv** |

Tabel Visualisasi *Queue* Proses BFS

Setelah queue terisi dengan proses *input*, maka aplikasi akan berjalan dengan mengeluarkan proses terdepan yang ada dalam *queue*. Setelah proses selesai dijalankan, jika proses menghasilkan data, maka data tersebut akan masuk kedalam sebuah list yang disebut *shared\_resource*. Data yang masuk kedalam *shared\_resource* memiliki *count* berapa kali data ini akan digunakan sesuai dengan kaki *output* proses pada *node* DAG. Pada kasus Gambar 4.3.1 proses *load hr-1.csv* memiliki alur menuju proses *append*, maka *metadata* proses *append* dimasukan kedalam *queue* dengan menambahkan juga *id* unik dari *shared\_list* yang nantinya akan digunakan proses selanjutnya. Terakhir proses *load hr-1.csv* dikeluarkan karena sudah selesai diproses

|  |  |
| --- | --- |
| *shared\_resource* | |
| 0:{data: *Data Load hr-1.csv*, count:1} |  |

|  |  |
| --- | --- |
| ***Load hr-2.csv*** | ***Append*** |

Tabel Visualisasi *shared\_resource* DAG

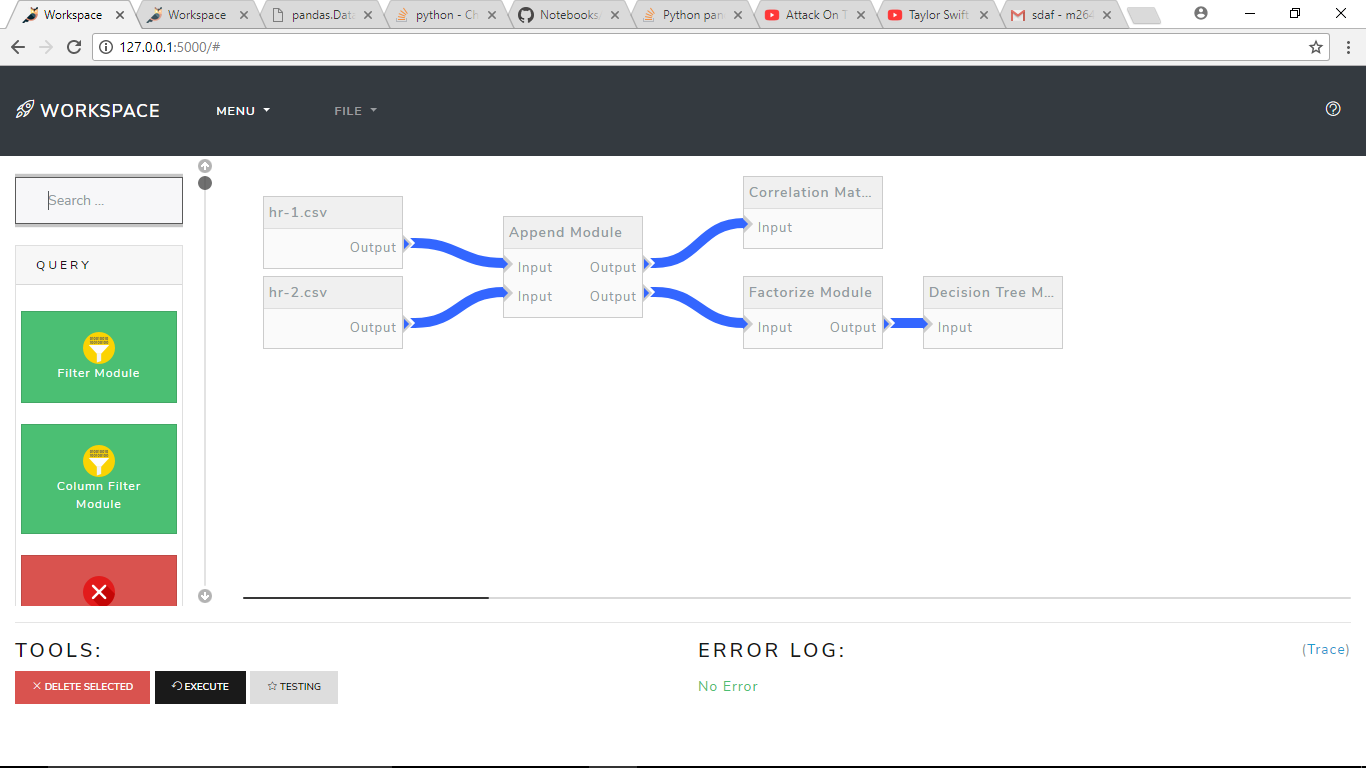
Proses akan terus bejalan sampai queue tidak memiliki isi lagi. Setiap data pada *shared\_resource* memiliki count yang sesuai dengan jumlah koneksi yang dimiliki node tersebut. Setiap kali data digunakan oleh proses lain, maka count akan berkurang 1 dan jika count sampai angka 0 maka data tersebut akan dihapus dari *shared\_resource*.

Setiap sirkulasi data yang terjadi, terdapat sebuah variable yang menyimpan data terakhir yang masuk ke dalam shared\_resource. Jika ketika proses selesai dijalankan namun shared\_resource tidak memiliki data yang tersisa, maka aplikasi akan menggunakan data yang terakhir masuk kedalam sistem.

Implementasi DAG

|  |
| --- |
| **def** run(self):  self.process\_init()  **while** len(self.process) > 0:  current = self.process[0]  **if** current[**'type'**] == **'input'**:  *#* ***Todo Change Input Condition*** df = pd.read\_csv(**"dummy/"** + str(current[**'name'**]))  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  tmp = {  **'data'**: df,  **'count'**: count  }  self.shared\_resource[self.id] = tmp  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:append'**:  **if** len(current[**'shared\_input\_resource'**]) == 2:  left, right = self.extract\_input(current, 2)  df = left.append(right)  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **else**:  self.process.append(current)  **elif** current[**'type'**] == **'process:join'**:  **if** len(current[**'shared\_input\_resource'**]) == 2:  left, right = self.extract\_input(current, 2)   df = left.merge(right, left\_on=current[**'metadata'**][**'left'**], right\_on=current[**'metadata'**][**'right'**], how=current[**'metadata'**][**'how'**])  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **else**:  self.process.append(current)  **elif** current[**'type'**] == **'process:cfilter'**:  input = self.extract\_input(current, 1)  desired\_column = []  **for** key **in** current[**'shape'**]:  desired\_column.append(key)  df = input[desired\_column]  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:filter'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  tools.set\_condition(current[**'query'**])  df = tools.get\_filter()  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:update-value'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  tools.set\_condition(current[**'query'**])  into = current[**'into'**]  **if** input[current[**'target'**]].dtype == **'int64'**:  into = int(into)  **elif** input[current[**'target'**]].dtype == **'float64'**:  into = float(into)  **elif** input[current[**'target'**]].dtype == **'object'**:  into = str(into)  tools.update(current[**'target'**], into)  df = tools.data\_frame()  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:update-column'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  tools.set\_condition(current[**'query'**])  into = input[current[**'into'**]]  tools.update(current[**'target'**], into)  df = tools.data\_frame()  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:delete'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  tools.set\_condition(current[**'query'**])  tools.delete()  df = tools.data\_frame()  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:aggregate'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  gb = []  **if ',' in** str(current[**'group\_by'**]).strip():  gb = str(current[**'group\_by'**]).strip().split(**','**)  **else**:  gb.append(str(current[**'group\_by'**]))  df = tools.get\_aggregate(gb, current[**'function'**], current[**'target'**])  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:formula'**:  tools = qt.QueryTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  tools.execute\_formula(current[**'formula'**], current[**'new\_name'**])  df = tools.data\_frame()  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:factorize'**:  input = self.extract\_input(current, 1)  df = input.copy()  df[current[**'target'**]] = pd.factorize(input[current[**'target'**]])[0]  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:fillna-aggregate'**:  input = self.extract\_input(current, 1)  f = current[**'function'**]  df = pd.DataFrame()  **if** f == **'sum'**:  df = input.fillna(input[current[**'target'**]].sum())  **elif** f == **'avg'**:  df = input.fillna(input[current[**'target'**]].mean())  **elif** f == **'count'**:  df = input.fillna(input[current[**'target'**]].count())  **elif** f == **'std'**:  df = input.fillna(input[current[**'target'**]].std())  **elif** f == **'max'**:  df = input.fillna(input[current[**'target'**]].max())  **elif** f == **'min'**:  df = input.fillna(input[current[**'target'**]].min())  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:fillna-oc'**:  input = self.extract\_input(current, 1)  df = input.copy()  df[current[**'target'**]].fillna(df[current[**'other'**]])  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'process:fillna-value'**:  input = self.extract\_input(current, 1)  df = input.copy()  into = df[current[**'value'**]]  **if** input.dtype[current[**'target'**]] == **'int64'**:  into = int(into)  **elif** input.dtype[current[**'target'**]] == **'float64'**:  into = float(into)  **elif** input.dtype[current[**'target'**]] == **'object'**:  into = str(into)  df[current[**'target'**]].fillna(into)  count = 1  **if** len(current[**'link'**]) > 0:  count = len(current[**'link'**])  self.shared\_resource[self.id] = {  **'data'**: df,  **'count'**: count  }  self.generate\_next\_bfs(current)  self.id += 1  **elif** current[**'type'**] == **'chart:cm'**:  tools = ct.ChartTools()  input = self.extract\_input(current, 1)  tools.set\_dataset(input.copy())  title = **"Correlation Matrix"** img = tools.corr\_matrix\_chart(title)  data = {  **'title'**: title,  **'img'**: img  }  self.chart.append(data)  **elif** current[**'type'**] == **'model:dt'**:  input = self.extract\_input(current, 1)  clf = tree.DecisionTreeClassifier()  x = input.drop(current[**'target'**], axis=1)  y = input[current[**'target'**]]  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state=42)  summary = y\_test.value\_counts()  support = {}  total\_data = 0  **for** key, value **in** summary.iteritems():  support[key] = value  total\_data += int(value)  clf = tree.DecisionTreeClassifier()  start = time.clock()  dt = clf.fit(x\_train, y\_train)  end = time.clock()  score = dt.score(x\_test, y\_test)  support\_table = pd.DataFrame({**"real"**: y\_test, **"predict"**: clf.predict(x\_test)})  support\_table[**'correct'**] = support\_table[**'predict'**] == support\_table[**'real'**]  support\_table[**'correct'**] = support\_table[**'correct'**].apply(int)  support\_metadata = {}  **for** key, value **in** summary.iteritems():  tmp = support\_table.groupby(**'real'**).sum()[**'correct'**][key]  d = {  **'count'**: float(value),  **'conf'**: float(tmp),  **'psupport'**: float(value) / total\_data \* 100,   **'pconf'**: float(tmp) / value \* 100   }  support\_metadata[key] = d  cv = cross\_val\_score(dt, x, y, cv=10)  performance = cv  objects = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  y\_pos = np.arange(len(objects))  plt.clf()  plt.bar(y\_pos, performance, align=**'center'**, alpha=0.5)  plt.xticks(y\_pos, objects)  plt.ylabel(**'Accuracy'**)  plt.xlabel(**'Fold'**)  plt.title(**'Decision Tree Cross Validation, 10 Fold'**)  tools = ct.ChartTools()  p = tools.convert\_base64(plt)  res = {  **"name"**: **"Decision Tree"**,  **"cv"**: cv.tolist(),  **"accuracy"**: float(cv.mean()),  **"error"**: float(cv.std() \* 2),  **"time"**: end - start,  **"support"**: support\_metadata,  **"score"**: score,  **"cv\_plot"**: p,  **"total\_test\_data"**: int(total\_data)  }  self.model.append(res)  self.process.pop(0)  **if** len(self.shared\_resource) == 0:  self.shared\_resource[self.id] = self.last\_resource  self.id += 1 |

Gambar Interface dari DAG yang dibuat:



Implementasi *Directed Acyclic Graph*